

Rational Application Developer



Guía de consulta de EGL

Versión 6 Release 001

Rational Application Developer



Guía de consulta de EGL

Versión 6 Release 001

Nota

Antes de utilizar esta información y el producto al que da soporte, lea la información de "Avisos", en la página 1077.

Tercera edición (abril de 2005)

Esta edición es aplicable a la versión 6, release 0, modificación 0 de Rational Web Developer y Rational Application Developer y a todos los releases y modificaciones subsiguientes hasta que se indique lo contrario en nuevas ediciones.

© Copyright International Business Machines Corporation 1996, 2005. Reservados todos los derechos.

Contenido

Visión general	1
Introducción a EGL	1
Novedades de EGL 6.0.0.1	1
Novedades del iFix de EGL 6.0	3
Novedades de EGL Versión 6.0	4
Proceso de desarrollo	8
Configuraciones de tiempo de ejecución	9
Utilización de una envoltura Java	10
Llamadas válidas	10
Transferencias válidas	11
Fuentes de información adicional acerca de EGL	12
Visión general del lenguaje EGL.	13
Proyectos, paquetes y archivos EGL	13
Proyecto EGL.	13
Paquete.	14
Archivos EGL	15
Recomendaciones	15
Componentes.	17
Referencias a componentes	21
Estructura fija	26
Typedef.	27
Import	33
Información general	33
Formato de la sentencia import.	33
Tipos primitivos.	34
Tipos primitivos durante la declaración	36
Eficacia relativa de los diversos tipos numéricos	36
ANY	37
Tipos de caracteres	38
Tipos de fecha y hora	41
Tipos LOB.	48
Tipos numéricos.	50
Declarar variables y constantes en EGL	53
Acceso dinámico y estático	54
Reglas de ámbito y "this" en EGL	56
Referencias a variables en EGL	58
Sintaxis de corchete para el acceso dinámico	60
Sintaxis abreviada para referirse a estructuras fijas	62
Visión general de las propiedades de EGL	64
Propiedades de presentación de campos.	66
Propiedades de formato	66
Propiedades de elementos SQL.	67
Propiedades de validación	67
Bloques de establecimiento de valor	67
Bloques de establecimiento de valor para situaciones elementales	68
Bloques de establecimiento de valor para un campo de un campo	69
Utilización de "this"	71
Bloques de establecimiento de valor, matrices y elementos de matriz	71
Ejemplos adicionales	72
Matrices	74

Matrices dinámicas	74
Matrices de campos de estructura	78
Diccionario	82
Propiedades de diccionario	84
Funciones de diccionario	85
ArrayDictionary	86
Sentencias EGL	88
Palabras clave por orden alfabético	91
Transferencia de control entre programas	93
Manejo de excepciones	94
Bloques try	95
Excepciones del sistema de EGL	95
Límites de los bloques try	96
Variables de sistema relacionadas con errores	97
Sentencias de E/S	98
Identificación de errores	99

Migrar código EGL a EGL 6.0 iFix	101
Migración de EGL a EGL	102
Cambios en las propiedades durante la migración de EGL a EGL	105
Establecer las preferencias de migración de EGL a EGL	110

Configurar el entorno	113
Establecer preferencias de EGL	113
Establecer las preferencias del texto	113
Establecer preferencias para el depurador de EGL	114
Establecer los descriptores de construcción por omisión	116
Establecer preferencias para el editor de EGL	117
Establecer preferencias para estilos de fuente	117
Establecer preferencias para plantillas	118
Establecer preferencias para conexiones a bases de datos SQL	119
Establecer preferencias para la recuperación de SQL	121
Habilitar posibilidades de EGL	122

Iniciar el desarrollo de código	125
Crear un proyecto	125
Crear un proyecto EGL	125
Crear un proyecto Web EGL	126
Especificar opciones de base de datos durante la creación del proyecto.	127
Crear una carpeta fuente de EGL.	127
Crear un paquete de EGL	128
Crear un archivo fuente EGL	128
Utilizar las plantillas EGL con la ayuda de contenido.	129
Accesos directos para EGL	129

Desarrollar código fuente EGL básico	131
Crear un componente EGL dataItem.	131

Componente dataItem	131
Crear un componente de registro EGL	132
Componentes de registro	132
Componentes de registro fijo	133
Tipos de registros y propiedades	135
Crear un componente de programa de EGL	138
Componente de programa	139
Crear un componente de función de EGL	140
Componente de función	140
Crear un componente de biblioteca de EGL	141
Componente de biblioteca de tipo basicLibrary	142
Componente de biblioteca de tipo nativeLibrary	143
Crear un componente dataTable de EGL	145
DataTable	146

Insertar fragmentos de código en archivos EGL y JSP 149

Establecer el foco en un campo de formulario	150
Probar navegadores para una variable de sesión	150
Recuperar el valor de una fila pulsada en una tabla de datos	151
Actualizar una fila en una tabla relacional	152

Trabajar con formularios de texto e impresión 153

Crear un componente formGroup de EGL	153
Componente FormGroup	153
Componente de formulario	154
Crear un formulario de impresión de EGL	155
Crear un formulario de texto de EGL	158
Visión general del editor de formularios de EGL	164
Editar grupos de formularios con el editor de formularios de EGL	164
Crear un filtro	166
Crear un formulario en el editor de formularios de EGL	166
Crear un campo de longitud constante	167
Crear un campo de longitud variable en un formulario de texto o impresión	168
Establecer preferencias de entrada paleta editor formularios EGL	170
Plantillas de formulario en el editor de formularios de EGL	171
Opciones de visualización del editor de formularios de EGL	174
Establecer preferencias para el editor de formularios de EGL	175
Filtros de formulario en el editor de formularios de EGL	176

Crear una interfaz de usuario de consola 177

Interfaz de usuario de consola	177
Crear una interfaz con consoleUI	178
Componentes de ConsoleUI y variables relacionadas	180
Window	180
Prompt	181
ConsoleField	181
ConsoleForm	181

Utilización de new en ConsoleUI	183
Opciones de pantalla ConsoleUI para UNIX	184

Crear una aplicación Web de EGL . . . 187

Soporte Web	187
Crear una aplicación Web EGL de tabla única	187
Asistente Páginas y componentes de datos de EGL	187
Crear una aplicación Web EGL de tabla única	188
Definir páginas Web en el asistente Páginas y componentes de datos EGL	190
Crear un componente EGL pageHandler	191
Soporte de Page Designer para EGL	192
PageHandler	194
Controles de JavaServer Faces y EGL	198
Crear un campo de EGL y asociarlo con un JSP Faces	199
Asociar un registro EGL con un JSP Faces	200
Enlazar componente mandato JavaServer Faces con PageHandler EGL	201
Utilizar la vista Edición rápida para el código de PageHandler	202
Enlazar componente entrada/salida JavaServer Faces con PageHandler EGL	203
Enlazar comp. recuadro selección JavaServer Faces con PageHandler EGL	203
Enlazar componente selección única JavaServer Faces con PageHandler EGL	204
Enlazar comp. selección múltiple JavaServer Faces con PageHandler EGL	206

Crear informes de EGL 209

Visión general de los informes de EGL	209
Visión general del proceso de creación de informes de EGL	210
Orígenes de datos	212
Registros de datos en la biblioteca	212
Manejador de informes de EGL	213
Funciones de manejador de informes predefinidas	214
Funciones de manejador de informes de EGL adicionales	215
Tipos de datos en documentos de diseño XML	216
Código de ejemplo para funciones de controlador de informes de EGL	217
Añadir un documento de diseño a un paquete	219
Utilizar plantillas de informe	220
Crear un manejador de informes de EGL	221
Crear manualmente un manejador de informes de EGL	221
Escribir código para controlar un informe	225
Generar archivos para un informe y ejecutarlo	227
Exportar informes	228

Trabajar con archivos y bases de datos 229

Soporte de SQL	229
SQL y sentencias EGL	229
Proceso del conjunto de resultados	233
Registros SQL y sus usos	235
Acceso a base de datos durante la declaración	239

SQL dinámico	240
Ejemplos de SQL	241
Base de datos por omisión	251
Informix y EGL	252
Tareas específicas de SQL	252
Recuperar datos de tabla SQL	252
Crear componentes dataItem de componente de reg. SQL (visión general)	253
Crear componentes de datos de EGL a partir de tablas de bases de datos relacionales	255
Ver la sentencia SQL SELECT para un registro SQL	258
Validar la sentencia SQL SELECT para un registro SQL	259
Construir una sentencia EGL prepare	259
Construir una sentencia SQL explícita a partir de una implícita	260
Restablecer una sentencia SQL explícita	261
Eliminar una sentencia SQL de una sentencia EGL relacionada con SQL	262
Resolver una referencia para visualizar una sentencia SQL implícita	262
Cómo se realiza una conexión JDBC estándar	263
Soporte de VSAM	264
Requisitos previos de acceso	264
Nombre de sistema	264
Soporte de MQSeries	265
Conexiones	265
Incluir mensaje en transacción	266
Personalización	266
Palabras clave EGL relacionadas con MQSeries	268
Llamadas directas a MQSeries	270
Mantener código EGL	275
Línea de comentario de código fuente EGL	275
Buscar componentes	275
Ver referencias de componente	276
Abrir un componente en un archivo .egl	277
Localizar un archivo fuente EGL en el Explorador de proyectos	278
Suprimir un archivo de EGL en el Explorador de proyectos	278
Depurar código EGL	279
Depurador EGL	279
Mandatos del depurador	279
Utilización de descriptores de construcción	282
Acceso a base de datos SQL	283
Sentencia call	283
Tipo de sistema utilizado durante la depuración	284
Puerto del depurador EGL	284
Recomendaciones	284
Depurar aplicaciones que no sean J2EE	285
Iniciar una aplicación no de J2EE en el depurador de EGL	285
Crear una configuración de lanzamiento en el depurador de EGL	286
Crear una configuración de lanzamiento de escucha de EGL	287
Depurar aplicaciones J2EE	288

Preparar un servidor para la depuración Web EGL	288
Iniciar un servidor para la depuración Web EGL	288
Iniciar una sesión de depuración Web EGL	289
Utilizar puntos de interrupción en el depurador de EGL	290
Recorrer una aplicación en el depurador de EGL	291
Ver variables en el depurador de EGL	292

Trabajar con componentes de construcción de EGL 293

Crear un archivo de construcción	293
Configurar opciones de construcción generales	293
Configurar asociaciones de archivo externo, impresora y colas	304
Configurar opciones de llamada y transferencia	311
Configurar referencias a otros archivos de construcción de EGL	319
Editar un vía de acceso de construcción de EGL	320

Generar, prepara y ejecutar salida de EGL 323

Generación	323
Generación de código Java en un proyecto	323
Construcción	326
Construir la salida de EGL	327
Plan de construcción	327
Programa Java, PageHandler y biblioteca	328
Archivo de resultados	328
Generar en el entorno de trabajo	329
Generación en el entorno de trabajo	330
Generar desde la interfaz por lotes del entorno de trabajo	331
Generación a partir de la interfaz por lotes del entorno de trabajo	332
Generar a partir del SDK (Software Development Kit) de EGL	333
Generación a partir del SDK (Software Development Kit) de EGL	333
Invocar un plan de construcción tras la generación	334
Generar Java; varios temas	335
Procesar código Java generado en un directorio	335
Generar código de despliegue para proyectos EJB	338
Establecer la variable EGL_GENERATORS_PLUGINDIR	339
Ejecutar código Java generado por EGL en la máquina local	339
Iniciar una aplicación Java básica o de interfaz de usuario de texto en el sistema local	339
Iniciar una aplicación Web en el sistema local	340
Script de construcción	342
Script de construcción Java	342
Servidor de construcción	343
Iniciar un servidor de construcción en AIX, Linux o Windows 2000/NT/XP	343

Desplegar salida Java generada por EGL 347

Propiedades de tiempo de ejecución Java	347
---	-----

En un entorno J2EE	347	Formato de un archivo de construcción EGL	380
En un entorno Java no J2EE	347	Opciones del descriptor de construcción	382
Descriptores de construcción y propiedades de programa	349	Scripts de construcción	406
Para obtener más información	349	Opciones necesarias en los scripts de construcción de EGL	406
Configuración del entorno de tiempo de ejecución no J2EE para código generado por EGL	350	Elemento callLink	407
Archivo de propiedades del programa	350	Si el tipo de callLink es localCall (valor por omisión)	407
Desplegar aplicaciones de Java fuera de J2EE	350	Si el tipo de callLink es remoteCall	407
Instalar el código de ejecución de EGL para Java	351	Si el tipo de callLink es.ejbCall	408
Incluir archivos JAR en la variable CLASSPATH del sistema destino	352	Propiedad alias del elemento callLink	409
Preparación de la biblioteca de cursos UNIX		Propiedad conversionTable del elemento callLink	409
cursos para el entorno de ejecución de EGL	352	Propiedad ctgKeyStore del elemento callLink	410
Configurar el escucha TCP/IP para una aplicación no J2EE llamada	353	Propiedad ctgKeyStorePassword del elemento callLink	410
Configurar entorno de ejecución J2EE para código generado por EGL.	354	Propiedad ctgLocation del elemento callLink	410
Eliminar archivos jar duplicados	355	Propiedad ctgPort del elemento callLink	411
Establecer valores de descriptor de despliegue	355	Propiedad JavaWrapper del elemento callLink	411
Actualizar el archivo de entorno J2EE	356	Propiedad linkType del elemento callLink	412
Actualizar el descriptor de despliegue manualmente	357	Propiedad library del elemento callLink	412
Establecer el nombre JNDI para proyectos EJB	358	Propiedad location del elemento callLink	413
Configuración del servidor J2EE para llamadas CICSJ2C	358	Propiedad luwControl del elemento callLink	414
Config. escucha TCP/IP para aplic. llamada en módulo cliente aplic. J2EE	359	Propiedad package del elemento callLink	415
Establecer una conexión JDBC J2EE	362	Propiedad parmForm del elemento callLink	416
Desplegar un archivo de propiedades de enlace	364	Propiedad pgmName del elemento callLink	417
Proporcionar acceso a archivos jar no de EGL	365	providerURL en elemento callLink	417
Consulta de EGL	369	Propiedad refreshScreen del elemento callLink	418
Compatibilidad de asignación en EGL	369	Propiedad remoteBind del elemento callLink	418
Asignación en diversos tipos numéricos	370	Propiedad remoteComType del elemento callLink	419
Otras asignaciones de tipos cruzados	370	Propiedad remotePgmType del elemento callLink	421
Relleno y truncamiento en tipos de caracteres	371	Propiedad serverID del elemento callLink	422
Asignación entre indicaciones de la hora	372	Propiedad type del elemento callLink	423
Asignación hacia o desde elementos subestructurados de estructuras fijas	373	Funciones C con EGL	424
Asignación de un registro fijo	373	Funciones BIGINT para C	427
Asignaciones	374	Tipos de datos C y tipos primitivos EGL	428
Elementos de asociación.	375	Funciones DATE para C.	429
commit	375	Funciones DATETIME e INTERVAL para C	430
conversionTable	375	Funciones DECIMAL para C	431
fileType	375	Invocar una función C desde un programa EGL	432
fileName	376	Funciones de pila para C	434
formFeedOnClose	376	Funciones de devolución para C	437
replace	376	Comentarios.	438
system	376	Compatibilidad con VisualAge Generator	439
systemName	377	ConsoleUI	441
text.	377	Propiedades y campos de ConsoleField.	441
Elemento asynchLink.	377	Propiedades de ConsoleForm en consoleUI de EGL	454
Archivo csouidpwd.properties para llamadas remotas	378	Campos de Menu en consoleUI de EGL	455
Propiedad package del elemento asynchLink	379	Campos de MenuItem en consoleUI de EGL	456
Propiedad recordName del elemento asynchLink	379	Campos de PresentationAttributes en consoleUI de EGL	458
Componente de registro básico en formato fuente EGL	379	Campos de Prompt en consoleUI de EGL	460
Componentes de construcción	380	Campos de Window en consoleUI de EGL	461
		containerContextDependent	465
		Autorización de base de datos y nombres de tabla	466
		Conversión de datos	467
		Conversión de datos cuando el invocante es código Java	468

Algoritmo de conversión	469	Visión general de la utilización de las clases de envoltura	552
Texto de idioma bidireccional	470	La clase de envoltura de programa	553
Inicialización de datos	471	El conjunto de clases de envoltura de parámetro	555
Componente DataItem en formato fuente EGL	472	El conjunto de clases de envoltura de matriz de elementos subestructurada	556
Componente DataTable en formato fuente EGL	473	Clases de envoltura de matriz dinámica	557
Vía de acceso de construcción EGL y eglpath.	477	Convenios de denominación de las clases de envoltura Java	558
EGLCMD.	478	Referencias cruzadas de tipos de datos	559
Sintaxis	478	Requisitos de controlador JDBC en EGL	560
Ejemplos	480	Palabra clave	561
Archivo de mandatos EGL	481	add.	561
Ejemplos de archivos de mandatos	482	call	563
Editor EGL	483	case	566
Ayuda de contenido en EGL	483	close	568
Enumeraciones en EGL	484	continue	570
Palabras reservadas EGL	486	converse	570
Palabras reservadas fuera de una sentencia SQL	486	delete	571
EGLSDK	488	display	573
Sintaxis	488	execute	574
Ejemplos	489	exit	578
Formato del archivo eglmaster.properties	490	for	580
Formato fuente EGL	491	forEach	581
Excepciones del sistema de EGL	492	forward	583
Límites de sistema EGL	494	freeSQL	584
Expresiones	495	get	585
Expresiones de fecha y hora	496	get absolute	591
Expresiones lógicas	497	get current	593
Expresiones numéricas	504	get first	594
Expresiones de texto	505	get last	595
Formato del archivo plugin.xml del descriptor de construcción maestro	506	get next	597
Componente FormGroup en formato fuente EGL	508	get previous	602
Propiedades de un área flotante de pantalla	510	get relative	606
Propiedades de un área flotante de impresión	510	goTo	608
Componente de formulario en formato fuente EGL	511	if, else	608
Propiedades de formulario de texto	513	move	610
Propiedades de formulario de impresión	513	open	616
Campos de formulario	514	openUI	620
Propiedades del campo de formulario de texto	515	prepare	630
Invocaciones de función	518	print	632
Variables de función	520	replace	632
Parámetros de función	522	return	635
Implicaciones de inOut y los modificadores relacionados	525	set	636
Componente de función en formato fuente EGL	527	show	646
Salida generada	529	transfer	646
Salida generada (referencia)	530	try	648
Vista Resultados de la generación	532	while	648
operador in	532	Biblioteca (salida generada).	649
Ejemplos con una matriz unidimensional	533	Componente de biblioteca en formato fuente EGL	649
Ejemplos con una matriz multidimensional	534	Operador like	656
Componente de registro indexado en formato fuente EGL	535	Archivo de propiedades de enlace (detalles)	657
Valores de error de E/S	536	Cómo se identifica el archivo de propiedades de enlace durante la ejecución	657
duplicate	537	Formato del archivo de propiedades de enlace	657
endOfFile.	537	Operador matches.	660
format.	538	Personalización de mensajes para el tiempo de ejecución de Java EGL	661
noRecordFound	538	Componente de registro MQ en formato fuente EGL	662
unique	539	Propiedades de registros MQ	665
Operador isa	539		
Propiedades de ejecución de Java (detalles)	540		
Clases de envoltura Java	551		

Nombre de cola	665	sqlDataCode.	713
Incluir mensaje en transacción.	665	sqlVariableLen	714
Abrir cola de entrada para uso exclusivo	665	timeFormat	715
Registros de opciones para registros MQ	665	timeStampFormat	716
Creación de alias de nombres	667	typeChkMsgKey	716
Cambios en identificadores EGL de archivos JSP		upperCase	717
y beans Java generados	668	validationOrder	717
Cómo se crean alias de los nombres Java	669	validatorDataTable	718
Cómo se crean alias de los nombres de		validatorDataTableMsgKey	719
envoltura Java	670	validatorFunction	719
Convenios de denominación	672	validatorFunctionMsgKey	720
Operadores y precedencia	673	validValues	720
Salida de la generación de programa Java	675	validValuesMsgKey	722
Salida de la generación de envoltura Java	676	value	722
Ejemplo	678	zeroFormat	722
Componente PageHandler en formato fuente EGL	679	Datos de programa aparte de los parámetros	723
Propiedades del componente PageHandler	682	Parámetros de programa	726
Propiedades del campo PageHandler	685	Componente de programa en formato fuente EGL	728
pfKeyEquate	686	Programa básico en formato fuente EGL	729
Propiedades a nivel de campo primitivo	686	Programa de UI de texto en formato fuente EGL	731
action	690	Propiedades de componente de programa	733
align	690	Formulario de entrada	736
byPassValidation	691	Registro de entrada	736
color	692	Referencias cruzadas de tipo de registro y tipo de	
column	693	archivo	737
currency	694	Propiedades que dan soporte a registros de	
currencySymbol	694	longitud variable	737
dateFormat	695	Registros de longitud variable con la propiedad	
displayName	697	lengthItem	737
displayUse	698	Registros de longitud variable con la propiedad	
fieldLen	698	numElementsItem	738
fill	699	Registros de longitud variable con las	
fillCharacter	699	propiedades lengthItem y numElementsItem	739
help	699	Registros de longitud variable pasados en una	
highlight	700	llamada o transferencia	739
inputRequired	700	Compatibilidad de referencia en EGL	739
inputRequiredMsgKey	700	Componente de registro relativo en formato fuente	
intensity	701	EGL	740
isBoolean	701	Unidad de ejecución	742
isDecimalDigit	702	resultSetID	743
isHexDigit	702	Componente de registro serie en formato fuente	
isNullable	702	EGL	743
isReadOnly	703	Códigos de datos SQL y variables de lenguaje	
lineWrap	704	principal EGL	744
lowerCase	705	Columnas de longitud fija y variable	745
masked	705	Compatibilidad entre los tipos de datos SQL y	
maxLength	705	los tipos primitivos EGL.	745
minimumInput	706	VARCHAR, VARGRAPHIC y los tipos de datos	
minimumInputMsgKey	706	LONG relacionados	747
modified	706	DATE, TIME y TIMESTAMP	747
needsSOSI	707	Diseño interno de los registros SQL	747
newWindow.	707	Componente de registro SQL en formato fuente	
numElementsItem	708	EGL	748
numericSeparator	709	Elemento de estructura en el formato fuente de	
outline	709	EGL	752
pattern	710	Subseries	753
persistent.	710	Diagrama de sintaxis para funciones EGL	754
protect.	711	Diagrama de sintaxis para sentencias y mandatos	
selectFromListItem	711	EGL	755
selectType	712	Bibliotecas del sistema	757
sign	712	Biblioteca ConsoleLib de EGL	757

Biblioteca ConverseLib de EGL	788	Código de error de ejecución de Java EGL	
Biblioteca DateTimeLib de EGL	791	CSO7163E	965
Biblioteca J2EELib de EGL	802	Código de error de ejecución de Java EGL	
Biblioteca JavaLib de EGL	805	CSO7164E	965
Biblioteca LobLib de EGL	831	Código de error de ejecución de Java EGL	
Biblioteca MathLib de EGL	839	CSO7165E	965
recordName.resourceAssociation	859	Código de error de ejecución de Java EGL	
Biblioteca ReportLib de EGL	861	CSO7166E	966
Biblioteca StrLib de EGL	867	Código de error de ejecución de Java EGL	
Biblioteca SysLib de EGL	887	CSO7360E	966
Biblioteca VGLib de EGL	916	Código de error de ejecución de Java EGL	
Variables de sistema fuera de bibliotecas EGL	921	CSO7361E	966
ConverseVar.	921	Código de error de ejecución de Java EGL	
SysVar.	927	CSO7488E	967
VGVar.	940	Código de error de ejecución de Java EGL	
Elemento transferToTransaction	952	CSO7489E	967
Propiedad alias en elementos de enlace		Código de error de ejecución de Java EGL	
relacionados con la transferencia	953	CSO7610E	967
Propiedad externallyDefined del elemento		Código de error de ejecución de Java EGL	
transferToTransaction	953	CSO7620E	968
Declaración use.	954	Código de error de ejecución de Java EGL	
Información general	954	CSO7630E	968
En un componente de biblioteca o programa	955	Código de error de ejecución de Java EGL	
En un componente formGroup	957	CSO7640E	968
En un componente pageHandler	957	Código de error de ejecución de Java EGL	
		CSO7650E	968
		Código de error de ejecución de Java EGL	
		CSO7651E	969
		Código de error de ejecución de Java EGL	
		CSO7652E	970
		Código de error de ejecución de Java EGL	
		CSO7653E	970
		Código de error de ejecución de Java EGL	
		CSO7654E	971
		Código de error de ejecución de Java EGL	
		CSO7655E	971
		Código de error de ejecución de Java EGL	
		CSO7656E	972
		Código de error de ejecución de Java EGL	
		CSO7657E	972
		Código de error de ejecución de Java EGL	
		CSO7658E	973
		Código de error de ejecución de Java EGL	
		CSO7659E	973
		Código de error de ejecución de Java EGL	
		CSO7669E	974
		Código de error de ejecución de Java EGL	
		CSO7670E	974
		Código de error de ejecución de Java EGL	
		CSO7671E	974
		Código de error de ejecución de Java EGL	
		CSO7816E	975
		Código de error de ejecución de Java EGL	
		CSO7819E	975
		Código de error de ejecución de Java EGL	
		CSO7831E	975
		Código de error de ejecución de Java EGL	
		CSO7836E	976
		Código de error de ejecución de Java EGL	
		CSO7840E	976
Código de error de ejecución de Java			
EGL	959		
Código de error de ejecución de Java EGL			
CSO7000E	960		
Código de error de ejecución de Java EGL			
CSO7015E	961		
Código de error de ejecución de Java EGL			
CSO7016E	961		
Código de error de ejecución de Java EGL			
CSO7020E	962		
Código de error de ejecución de Java EGL			
CSO7021E	962		
Código de error de ejecución de Java EGL			
CSO7022E	962		
Código de error de ejecución de Java EGL			
CSO7023E	962		
Código de error de ejecución de Java EGL			
CSO7024E	963		
Código de error de ejecución de Java EGL			
CSO7026E	963		
Código de error de ejecución de Java EGL			
CSO7045E	963		
Código de error de ejecución de Java EGL			
CSO7050E	963		
Código de error de ejecución de Java EGL			
CSO7060E	964		
Código de error de ejecución de Java EGL			
CSO7080E	964		
Código de error de ejecución de Java EGL			
CSO7160E	964		
Código de error de ejecución de Java EGL			
CSO7161E	964		
Código de error de ejecución de Java EGL			
CSO7162E	965		

Código de error de ejecución de Java EGL	
VGJ1206E	1063
Código de error de ejecución de Java EGL	
VGJ1207E	1063
Código de error de ejecución de Java EGL	
VGJ1208E	1064
Código de error de ejecución de Java EGL	
VGJ1209E	1064
Código de error de entorno de ejecución Java EGL	
VGJ1210E	1064
Código de error de entorno de ejecución Java EGL	
VGJ1211E	1064
Código de error de ejecución de Java EGL	
VGJ1212E	1065
Código de error de ejecución de Java EGL	
VGJ1213E	1065
Código de error de ejecución de Java EGL	
VGJ1214E	1065
Código de error de ejecución de Java EGL	
VGJ1215E	1065
Código de error de ejecución de Java EGL	
VGJ1216E	1066
Código de error de ejecución de Java EGL	
VGJ1217W	1066
Código de error de ejecución de Java EGL	
VGJ1218E	1066
Código de error de ejecución de Java EGL	
VGJ1219E	1066
Código de error de ejecución de Java EGL	
VGJ1220E	1067
Código de error de ejecución de Java EGL	
VGJ1221E	1067
Código de error de ejecución de Java EGL	
VGJ1222E	1067
Código de error de ejecución de Java EGL	
VGJ1223E	1067
Código de error de ejecución de Java EGL	
VGJ1224E	1068
Código de error de ejecución de Java EGL	
VGJ1225E	1068
Código de error de ejecución de Java EGL	
VGJ1226E	1068
Código de error de ejecución de Java EGL	
VGJ1227E	1068
Código de error de ejecución de Java EGL	
VGJ1228E	1069
Código de error de ejecución de Java EGL	
VGJ1229E	1069
Código de error de ejecución de Java EGL	
VGJ1290E	1069
Código de error de ejecución de Java EGL	
VGJ1301E	1069
Código de error de ejecución de Java EGL	
VGJ1302E	1070

Código de error de ejecución de Java EGL	
VGJ1303E	1070
Código de error de ejecución de Java EGL	
VGJ1304E	1070
Código de error de ejecución de Java EGL	
VGJ1305E	1070
Código de error de ejecución de Java EGL	
VGJ1306E	1071
Código de error de ejecución de Java EGL	
VGJ1401E	1071
Código de error de ejecución de Java EGL	
VGJ1402E	1071
Código de error de ejecución de Java EGL	
VGJ1403E	1071
Código de error de ejecución de Java EGL	
VGJ1404E	1072
Código de error de ejecución de Java EGL	
VGJ1405E	1072
Código de error de ejecución de Java EGL	
VGJ1406E	1072
Código de error de ejecución de Java EGL	
VGJ1407E	1072
Código de error de ejecución de Java EGL	
VGJ1408E	1073
Código de error de ejecución de Java EGL	
VGJ1409E	1073
Código de error de ejecución de Java EGL	
VGJ1410E	1073
Código de error de ejecución de Java EGL	
VGJ1411E	1073
Código de error de ejecución de Java EGL	
VGJ1412E	1074
Código de error de ejecución de Java EGL	
VGJ1414E	1074
Código de error de ejecución de Java EGL	
VGJ1415E	1074
Código de error de ejecución de Java EGL	
VGJ1416E	1074
Código de error de ejecución de Java EGL	
VGJ1417E	1075
Código de error de ejecución de Java EGL	
VGJ1419E	1075
Código de error de ejecución de Java EGL	
VGJ9900E	1075
Código de error de ejecución de Java EGL	
VGJ9901E	1076

Apéndice. Avisos	1077
Información de interfaces de programación	1079
Marcas registradas y marcas de servicio	1079

Índice	1081
-------------------------	-------------

Visión general

Introducción a EGL

Lenguaje de generación para empresas (EGL) es un entorno de desarrollo y un lenguaje de programación que le permite escribir de forma rápida aplicaciones con funciones completas y centrar toda la atención en el problema empresarial que el código está tratando en lugar de concentrarse en las tecnologías de software. Por ejemplo, puede utilizar sentencias de E/S similares para acceder a diferentes tipos de almacenes de datos externos, tanto si estos almacenes de datos son archivos, bases de datos relacionales o colas de mensajes. Los detalles de Java y J2EE también permanecen ocultos para el usuario, de modo que se pueden presentar datos de empresa en los navegadores aunque se tenga poca experiencia en las tecnologías Web.

Después de codificar un programa EGL, génerele para crear el fuente Java ; a continuación, EGL prepara la salida para generar objetos ejecutables. EGL también puede proporcionar los siguientes servicios:

- Coloca el fuente en una plataforma de despliegue fuera de la plataforma de desarrollo
- Prepara el fuente en la plataforma de desarrollo
- Envía la información de estado desde la plataforma de despliegue a la plataforma de desarrollo, para que el usuario pueda comprobar los resultados

EGL incluso genera salida que facilita el despliegue final de los objetos ejecutables.

Un programa EGL escrito para una plataforma destino puede convertirse fácilmente para utilizarse en otra plataforma. La ventaja es que se puede codificar en función de los requisitos de la plataforma actual, y muchos detalles de una migración futura se manejan automáticamente. EGL también puede generar varios componentes de un sistema de aplicación a partir del mismo fuente.

Conceptos relacionados

“Proceso de desarrollo” en la página 8

“Proyectos, paquetes y archivos EGL” en la página 13

“Salida generada” en la página 529

“Componentes” en la página 17

“Configuraciones de tiempo de ejecución” en la página 9

Tareas relacionadas

“Crear un proyecto Web EGL” en la página 126

Consulta relacionada

“Editor EGL” en la página 483

“Formato fuente EGL” en la página 491

Novedades de EGL 6.0.0.1

La Versión 6.0.0.1 incluye los siguientes cambios:

- El editor de formularios EGL suministra una interfaz gráfica de usuario para crear formularios de texto e imprimibles.

- Los entornos destino incluyen HP-UX y Solaris. EGL suministra soporte de 32 y 64 bits para esas plataformas y ha añadido soporte de 64 bits para AIX.
- El depurador EGL ha experimentado los siguientes cambios:
 - Permite depurar aplicaciones basadas en la UI de consola
 - Permite utilizar una página de códigos EBCDIC para representar datos de caracteres y numéricos durante una sesión de depuración
- El lenguaje es más flexible:
 - Las variables de sistema **SysVar.sqlCode** y **SysVar.sqlState** son modificables
 - Los subíndices de matriz y los índices de subserie pueden incluir expresiones numéricas, siempre que estas expresiones no incluyan funciones
 - Las funciones que devuelven un valor pueden invocarse desde el interior de una expresión numérica, de texto o lógica si el tipo del valor de retorno es válido en la expresión
 - Las funciones que devuelven un valor pueden utilizarse como argumento para un parámetro de función que tenga el modificador **in**, si los tipos del valor de retorno y del parámetro son compatibles en cuanto a asignación
 - Las variables de sistema EGL pueden pasarse como argumento a cualquier parámetro de función que tenga el modificador **in**, si los tipos del argumento y del parámetro son compatibles en cuanto a asignación
 - Las variables de sistema EGL modificables pueden pasarse como argumento a un parámetro de función que tenga el modificador **out** (si los tipos del argumento y del parámetro son compatibles en cuanto a asignación) o **inOut** (si los tipos del argumento y del parámetro son compatibles en cuanto a referencia)
- La documentación identifica ahora el modificador de acceso (**in**, **out** o **inOut**) para todos los parámetros de todas las funciones de sistema EGL; y describe la compatibilidad de referencia y asignación
- Hay funciones de sistema nuevas disponibles:
 - **MathLib.stringAsDecimal** acepta un valor de carácter (como por ejemplo "98.6") y devuelve el valor equivalente de tipo DECIMAL.
 - **MathLib.stringAsFloat** acepta un valor de carácter (como por ejemplo "98.6") y devuelve el valor equivalente de tipo FLOAT.
 - **MathLib.stringAsInt** acepta un valor de carácter (como por ejemplo "98") y devuelve el valor equivalente de tipo BIGINT.
 - **SysLib.conditionAsInt** acepta una expresión lógica (como por ejemplo *myVar* == 6) , devolviendo un 1 si la expresión es true y un 0 si la expresión es false.
 - **SysLib.startLog** abre un archivo de anotaciones de error. El texto se escribe en dichas anotaciones cada vez que el programa invoca **SysLib.errorLog**.
 - **SysLib.errorLog** copia texto en las anotaciones de error que ha iniciado la función de sistema **SysLib.startLog**.
 - Nuevas funciones soportan la UI de consola:
 - **ConsoleLib.currentArrayCount** devuelve un número de elementos de la matriz dinámica que está asociada al formulario activo actual.
 - **ConsoleLib.setCurrentArrayCount** especifica el número de filas que existe en una matriz dinámica que está enlazada a un arrayDictionary en pantalla.
 - **ConsoleLib.hideAllMenuItems** oculta todas las opciones del menú que se visualiza actualmente
 - **ConsoleLib.showAllMenuItems** muestra todas las opciones del menú que se visualiza actualmente
- La herramienta de conversión Informix 4GL se incluye en el producto

- La herramienta de migración VAGen contiene cambios que permiten una migración más eficiente

Conceptos relacionados

“Fuentes de información adicional acerca de EGL” en la página 12

Novedades del iFix de EGL 6.0

Nota: EGL proporciona servicios para ayudarle a convertir el código antiguo en código que funciona con el iFix de EGL 6.0:

- Si utilizó una versión de EGL anterior a 6.0 para crear una aplicación Web basada en JavaServer Faces, haga lo siguiente en el entorno de trabajo:
 1. Pulse **Ayuda > Ayuda de Rational**
 2. En el recuadro de texto **Buscar** del sistema de ayuda, teclee como mínimo los caracteres iniciales de esta serie: *Migrar recursos de JavaServer Faces en un proyecto Web*
 3. Pulse Ir
 4. Pulse *Migrar recursos de JavaServer Faces en un proyecto Web* y siga las instrucciones de ese tema
- Para obtener más detalles acerca de cómo migrar código de EGL 6.0 o de una versión anterior, consulte la sección *Migrar código de EGL a EGL 6.0 iFix*.
- Si está migrando código de Informix 4GL o de VisualAge Generator, consulte *Fuentes de información adicional sobre EGL*.

La versión 6.0 iFix representa una actualización significativa del lenguaje EGL:

- Presenta el manejador de informes de EGL que contiene funciones personalizadas que se invocan varias veces durante la ejecución de un archivo de diseño JasperReports. Los datos devueltos por cada función se incluyen en el informe de salida, que puede representarse en formato PDF, XML, texto o HTML. La tecnología es una mejora de la posibilidad de creación de informes disponible en Informix 4GL.
- Presenta la UI de consola de EGL que es una tecnología para crear una interfaz basada en caracteres que permite una interacción inmediata, controlada por teclado entre el usuario y el programa Java generado por EGL. La tecnología es una mejora de la interfaz de usuario dinámica disponible en Informix 4GL.
- Proporciona flexibilidad para el desarrollo de código:
 - Permite declarar tipos nuevos de variables:
 - Una variable de referencia que no contiene datos comerciales pero que apunta a tales datos.
 - Una variable que contiene o hace referencia a una gran cantidad de datos; específicamente a un objeto grande binario (BLOB) o a un objeto grande de caracteres (CLOB).
 - Una variable de serie que hace referencia a una serie Unicode cuya longitud varía en tiempo de ejecución.
 - Una variable de tipo ANY que puede contener datos comerciales de cualquier tipo primitivo.
 - Permite incluir invocaciones de función en expresiones.
 - Permite hacer referencia a un registro sin tener conocimiento de desarrollo del tamaño o de otras características del registro o de los campos del registro. Cada campo puede hacer referencia a un registro.

- Amplía el soporte para matrices dinámicas que ahora puede tener varias dimensiones.
- Presenta dos clases nuevas de colecciones de datos:
 - Un diccionario compuesto de un conjunto de entradas de clave y valor. Puede añadir, suprimir y recuperar entradas en tiempo de ejecución y el valor de una entrada dada puede ser de cualquier tipo.
 - Un arrayDictionary compuesto de un conjunto de matrices unidimensionales, cada una de cualquier tipo. Puede acceder al contenido de un arrayDictionary recuperando de todas las matrices los elementos con el mismo número.
- Amplía el número de funciones del sistema con varios propósitos:
 - Para mejorar el proceso de fecha y hora, el manejo de mensajes de tiempo de ejecución y la recuperación de propiedades de tiempo de ejecución Java definidas por el usuario.
 - Para soportar la funcionalidad nueva relacionada con informes, UI de consola, BLOB y CLOB.
- Proporciona un soporte mejor para el manejo de excepciones, para la inicialización de datos y para el acceso a las DLL.
- Proporciona un asistente nuevo para crear manejadores de informes de EGL.
- Permite personalizar una plantilla de página Web para utilizarla con el asistente Páginas y componentes de datos, que proporciona rápidamente una aplicación Web para acceder a una sola base de datos relacional.
- Permite crear código que refleja el comportamiento en tiempo de ejecución de Informix 4GL en relación con el proceso nulo y los compromisos de base de datos.

Conceptos relacionados

“Migración de EGL a EGL” en la página 102

“Fuentes de información adicional acerca de EGL” en la página 12

“Novedades de EGL Versión 6.0”

Novedades de EGL Versión 6.0

Nota: Si utilizó una versión anterior de EGL para crear una aplicación Web basada en JavaServer Faces, haga lo siguiente en el entorno de trabajo:

1. Pulse **Ayuda > Ayuda de Rational**
2. En el recuadro de texto **Buscar** del sistema de ayuda, teclee como mínimo los caracteres iniciales de esta serie: *Migrar recursos de JavaServer Faces en un proyecto Web*
3. Pulse Ir
4. Pulse *Migrar recursos de JavaServer Faces en un proyecto Web* y siga las instrucciones de ese tema

La versión 6.0 aumenta la potencia del lenguaje EGL:

- El proceso de bases de datos relacionales ha mejorado
 - Asistentes nuevos que permiten llevar a cabo rápidamente lo siguiente:
 - Crear componentes de datos directamente de tablas de bases de datos relacionales
 - Crear aplicaciones Web que crean, leen actualizan y suprimen filas de esas tablas
 - Hay funciones de sistema nuevas disponibles:

- **sysLib.loadTable** carga información de un archivo y la inserta en una tabla de base de datos relacional.
- **sysLib.unloadTable** descarga información de una tabla de base de datos relacional y la inserta en un archivo.
- Si está generando código Java puede acceder a filas de base de datos SQL en un cursor navegando a la fila siguiente (como se ha hecho siempre), navegando a la primera o última fila o a la fila anterior o actual o especificando una posición absoluta o relativa en el cursor.
- La sentencia **forEach** permite recorrer fácilmente las filas de un conjunto de resultados de SQL.
- La sentencia **freeSQL** libera los recursos asociados a una sentencia SQL preparada dinámicamente, cerrando cualquier cursor abierto asociado con esa sentencia SQL.
- El proceso de series ha mejorado
 - Ahora puede especificar subseries en una expresión de texto como en el ejemplo siguiente:


```
myItem01 = "1234567890";

// myItem02 = "567"
myItem02 = myItem01[5:7];
```
 - Puede especificar un retroceso, una avance de hoja o una tabulación en un literal de texto
 - Puede comparar series con dos tipos de patrones:
 - Un patrón de tipo SQL que incluye la palabra clave LIKE. A continuación se ofrece un ejemplo:


```
// la variable myVar01 es la expresión de serie
// cuyo contenido se comparará con un criterio like
myVar01 = "abcdef";

// la expresión lógica siguiente evalúa a "true"
if (myVar01 like "a_c%")
;
end
```
 - Un patrón de expresión regular. A continuación se ofrece un ejemplo:


```
// la variable myVar01 es la expresión de serie
// cuyo contenido se comparará con un criterio match
myVar01 = "abcdef";

// la expresión lógica siguiente evalúa a "true"

if (myVar01 matches "a?c*")
;
end
```
 - Puede utilizar estas funciones de sistema de formato de texto:
 - strLib.characterAsInt**
Convierte una serie de caracteres en una serie de enteros
 - strLib.clip**
Suprime espacios en blanco finales y nulos del final de las series de caracteres devueltas
 - strLib.formatNumber**
Devuelve un número como serie con formato
 - strLib.integerAsChar**
Convierte una serie de entero en una serie de caracteres

strLib.lowercase

Convierte todos los valores en mayúsculas de una serie de caracteres en valores en minúsculas

strLib.spaces

Devuelve una serie de una longitud especificada.

strLib.upperCase

Convierte todos los valores en minúsculas de una serie de caracteres en valores en mayúsculas.

- Puede declarar variables y elementos de estructura de tipos nuevos.

Los tipos numéricos nuevos son los siguientes:

FLOAT

Hace referencia a un área de 8 bytes que almacena números de coma flotante de doble precisión con 16 dígitos significativos como máximo

MONEY

Hace referencia a una cantidad de moneda almacenada como número decimal de coma flotante de 32 dígitos significativos como máximo

SMALLFLOAT

Hace referencia a un área de 4 bytes que almacena un número de coma flotante de precisión simple con 8 dígitos significativos como máximo

Los tipos de fecha y hora nuevos son los siguientes:

DATE

Hace referencia a una fecha de calendario específica, representada en 8 dígitos de un solo byte

INTERVAL

Hace referencia a un intervalo de tiempo representado con entre 1 y 21 dígitos de un solo byte y está asociado con una máscara como por ejemplo "hhmmss" para horas, minutos y segundos

TIME

Hace referencia a una instancia de hora, representada en 6 dígitos de un solo byte

TIMESTAMP

Hace referencia a una instancia de hora representada con entre 1 y 20 dígitos de un solo byte y está asociada con una máscara como por ejemplo "aaaaMMddhh" para año, mes, día y hora

- La sintaxis proporciona opciones adicionales:

- Siempre puede hacer referencia a un elemento de una matriz de elemento de estructura de la manera siguiente, pero en el caso de cambios de iFix, deberá evitar esta sintaxis:

```
mySuperItem.mySubItem.mySubmostItem[4,3,1]
```

Se recomienda encarecidamente utilizar la sintaxis siguiente:

```
mySuperItem[4].mySubItem[3].mySubmostItem[1]
```

- Puede utilizar una lista de identificadores delimitada por comas cuando declara parámetros, entradas de sentencia use, entradas de sentencia set o variables como en este ejemplo:

```
myVariable01, myVariable02 myPart;
```

- En una expresión numérica, ahora puede especificar un exponente precediendo un valor con un asterisco doble (**), de forma que (por ejemplo) 8 al cubo es 8**3

- Ahora puede especificar expresiones que se resuelvan cada una en fecha, hora, indicación de la hora o intervalo y la aritmética de fechas permite hacer varias tareas, como por ejemplo calcular el número de minutos entre dos fechas
- Las adiciones siguientes también permiten el proceso de fecha y hora:
 - **DateTimeLib.currentTime** y **DateTimeLib.currentTimeStamp** son variables del sistema que reflejan la hora actual
 - Las funciones de formato nuevo están disponibles para fechas (**StrLib.formatDate**), horas (**StrLib.formatTime**) e indicaciones de la hora (**sysLib.TimeStamp**)
 - Cada una de las funciones siguientes permite convertir una serie de caracteres en un elemento de un tipo de fecha y hora de forma que el elemento pueda utilizarse en una expresión de fecha y hora:
 - **DateTimeLib.dateValue** devuelve una fecha
 - **DateTimeLib.timeValue** devuelve una hora
 - **DateTimeLib.timeStampValue** devuelve una indicación de la hora asociada con una máscara determinada como por ejemplo "aaaaMMdd"
 - **DateTimeLib.intervalValue** devuelve un intervalo asociado a una máscara determinada como por ejemplo, "aaaaMMdd"
 - **DateTimeLib.extendDateValue** acepta una fecha, hora o indicación de la hora y la amplía a un elemento asociado a una máscara determinada, como por ejemplo "aaaaMMddmmss"
- Puede utilizar estas sentencias generales nuevas:
 - La sentencia **for** incluye un bloque de sentencia que se ejecuta en un bucle tantas veces como una prueba de como resultado true. La prueba se realiza al principio del bucle e indica si el valor de un contador está dentro de un rango especificado.
 - La sentencia **continue** transfiere el control al final de una sentencia **for**, **forEach** o **while** que contiene la sentencia **continue**. La ejecución de la sentencia continúa o finaliza dependiendo de la prueba lógica realizada al inicio de la sentencia contenedora.
- Puede ejecutar un mandato del sistema de forma síncrona (emitiendo la función **sysLib.callCmd**) o asíncrona (emitiendo la función **sysLib.startCmd**).
- Puede utilizar dos funciones nuevas que le permiten acceder a los argumentos de línea de mandatos de un bucle
 - **sysLib.callCmdLineArgCount** devuelve el número de argumentos
 - **sysLib.callCmdLineArg** devuelve el argumento que reside en una posición especificada de la lista de argumentos
- Ahora puede especificar una sentencia **case** en la que cada cláusula estará asociada a una expresión lógica distinta. Si utiliza esta sintaxis nueva, el tiempo de ejecución de EGL ejecuta las sentencias asociadas con la primera expresión verdadera:

```

case
  when (myVar01 == myVar02)
    conclusion = "bien";
  when (myVar01 == myVar03)
    conclusion = "hay que investigar";
  otherwise
    conclusion = "mal";
end

```

- Puede controlar si un parámetro de función se utiliza solo para entrada, solo para salida o en ambos casos y puede evitar la opción aceptando el valor predeterminado, que es el valor no restringido "for both".
- Ahora puede especificar una expresión de fecha y hora, de texto o numérica que sea más compleja que un solo elemento o constante en los casos siguientes:
 - Cuando especifica el valor proporcionado al sistema operativo por una sentencia **return**
 - Cuando especifica un argumento que se pasa a una invocación de función o a una llamada de programa; sin embargo, las características del parámetro de recepción deben conocerse durante la generación
- Ahora puede especificar una expresión numérica compleja al salir del programa

El entorno de desarrollo también ha mejorado:

- Hay dos características nuevas que proporcionan la posibilidad de acceder rápidamente a componentes, incluso conforme va aumentando la complejidad del código:
 - La vista Referencia de componentes permite visualizar una lista jerárquica de los componentes de EGL a los que haga referencia un programa, una biblioteca o un PageHandler y, desde esa lista, puede acceder a cualquiera de los componentes a los que se hace referencia
 - El mecanismo de búsqueda de EGL permite especificar un criterio de búsqueda para acceder a un conjunto de componentes o variables en el área de trabajo o en un subconjunto de los proyectos
- Finalmente, la perspectiva Web de EGL se ha eliminado en favor de la perspectiva Web ampliamente utilizada.

Conceptos relacionados

"Migración de EGL a EGL" en la página 102

"Fuentes de información adicional acerca de EGL" en la página 12

"Novedades del iFix de EGL 6.0" en la página 3

Proceso de desarrollo

El trabajo con EGL incluye los pasos siguientes:

Configuración

Configure un entorno de trabajo; por ejemplo, establezca preferencias y cree proyectos.

Crear y abrir archivos EGL

Empiece a crear el código fuente.

Declaración

Cree y especifique los detalles del código.

Validación

En varios momentos (por ejemplo al guardar un archivo), EGL revisa las declaraciones e indica si éstas son sintácticamente correctas y (en cierta medida) si son coherentes internamente.

Depuración

Puede interactuar con un depurador incorporado para asegurar que el código cumple los requisitos.

Generación

EGL valida las declaraciones y crea la salida, incluido el código fuente.

Preparación

EGL prepara el código fuente para generar objetos ejecutables. En algunos casos, este paso coloca el código fuente en una plataforma de despliegue fuera de la plataforma de desarrollo, prepara el código fuente en la plataforma de despliegue y envía un archivo de resultados desde la plataforma de despliegue a la plataforma de desarrollo.

Ejecución

En algunos casos, puede ejecutar el código inmediatamente en el entorno de trabajo simplemente pulsando con el botón derecho del ratón en la salida Java y pulsando **Ejecutar > Aplicación Java**.

Despliegue

EGL genera la salida y facilita el despliegue de los objetos ejecutables.

Conceptos relacionados

“Depurador EGL” en la página 279

“Generación de código Java en un proyecto” en la página 323

“Introducción a EGL” en la página 1

Tareas relacionadas

“Procesar código Java generado en un directorio” en la página 335

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

Consulta relacionada

“Editor EGL” en la página 483

“Formato fuente EGL” en la página 491

Configuraciones de tiempo de ejecución

EGL le permite generar un programa Java para cualquiera de las diversas plataformas soportadas. Puede desplegar el programa fuera de J2EE o en el contexto de uno de los siguientes contenedores J2EE:

- Cliente de aplicaciones J2EE
- Aplicación Web J2EE
- Contenedor EJB; en este caso, también se genera un bean de sesión EJB

Además, EGL proporciona una forma de definir una aplicación Web que tiene las siguientes características:

- Proporciona páginas gráficas a navegadores Web
- Puede almacenar y recuperar datos para un número potencialmente grande de usuarios
- Está incorporado en una infraestructura basada en Java llamada JavaServer Faces

Para obtener información detallada sobre este soporte especializado para aplicaciones Web, consulte la sección *Componente PageHandler*.

Finalmente, puede utilizar EGL para generar una envoltura Java, como se describe en la siguiente sección.

Utilización de una envoltura Java

La envoltura Java generada por EGL es un conjunto de clases que permiten invocar un programa generado por EGL desde el código Java no generado por EGL; por ejemplo, desde una clase de acción de una aplicación Web J2EE basada en Struts o JSF o desde un programa Java no J2EE. La tarea de integración de Java con EGL es la siguiente:

1. Genere las clases de envoltura Java, que son específicas de un programa generado
2. Incorpore estas clases de envoltura en el código Java no generado
3. Desde el código Java no generado, invoque los métodos de clase de envoltura para realizar la llamada y convertir datos entre estos dos formatos:
 - Los formatos de tipo de datos que utiliza Java
 - Los formatos de tipo primitivo necesarios cuando se pasan datos a o desde el programa generado por EGL

Llamadas válidas

La tabla siguiente muestra las llamadas válidas a o desde el código generado por EGL.

Objeto llamante	Objeto llamado
Una envoltura Java generada por EGL de una clase Java que está fuera de J2EE	Un programa Java generado por EGL (no J2EE)
	Un programa Java generado por EGL de un cliente de aplicaciones J2EE
	Un bean de sesión EJB generado por EGL
	Un programa CICS COBOL que se ha generado mediante VisualAge Generator
Una envoltura Java generada por EGL de un cliente de aplicaciones J2EE	Un programa Java generado por EGL (no J2EE)
	Un programa Java generado por EGL de un cliente de aplicaciones J2EE
	Un bean de sesión EJB generado por EGL
	Un programa CICS COBOL que se ha generado mediante VisualAge Generator
Una envoltura Java generada por EGL de una aplicación Web J2EE	Un programa Java generado por EGL (no J2EE)
	Un programa Java generado por EGL de un cliente de aplicaciones J2EE
	Un programa Java generado por EGL de la misma aplicación Web J2EE
	Un bean de sesión EJB generado por EGL
	Un programa CICS COBOL que se ha generado mediante VisualAge Generator

Objeto llamante	Objeto llamado
Un programa Java generado por EGL que está fuera de J2EE	Un programa Java generado por EGL (no J2EE)
	Un programa Java generado por EGL de un cliente de aplicaciones J2EE
	Un bean de sesión EJB generado por EGL
	Un programa CICS COBOL que se ha generado mediante VisualAge Generator
	Un programa no generado por EGL que se ha escrito en C o C++
	Un programa no generado que se ha escrito en cualquier lenguaje y se ejecuta bajo CICS
Un programa Java generado por EGL que está en un cliente de aplicaciones J2EE	Un programa Java generado por EGL (no J2EE)
	Un programa Java generado por EGL de un cliente de aplicaciones J2EE
	Un bean de sesión EJB generado por EGL
	Un programa CICS COBOL generado por EGL
	Un programa no generado que se ha escrito en cualquier lenguaje y se ejecuta bajo CICS
	Un programa no generado que se ha escrito en C o C++
Un programa Java generado por EGL de una aplicación Web J2EE	Un programa Java generado por EGL (no J2EE)
	Un programa Java generado por EGL de un cliente de aplicaciones J2EE
	Un programa Java generado por EGL de la misma aplicación Web J2EE
	Un bean de sesión EJB generado por EGL
	Un programa CICS COBOL que se ha generado en VisualAge Generator
	Un programa no generado escrito en C o C++
Un bean de sesión EJB generado por EGL	Un programa Java generado por EGL (no J2EE)
	Un programa Java generado por EGL de un cliente de aplicaciones J2EE
	Un bean de sesión EJB generado por EGL
	Un programa CICS COBOL que se ha generado mediante VisualAge Generator
	Un programa no generado escrito en C o C++

Transferencias válidas

La tabla siguiente muestra las transferencias válidas a o desde el código generado por EGL.

Objeto que realiza la transferencia	Objeto receptor
Un programa Java generado por EGL que está fuera de J2EE	Un programa Java generado por EGL (no J2EE)
Un programa Java generado por EGL que está en un cliente de aplicaciones J2EE	Un programa Java generado por EGL del mismo cliente de aplicaciones J2EE
Un programa Java generado por EGL de una aplicación Web J2EE	Un programa Java generado por EGL de la misma aplicación Web J2EE

Conceptos relacionados

“Salida generada” en la página 529

“Introducción a EGL” en la página 1

“Programa Java, PageHandler y biblioteca” en la página 328

“Envoltura Java” en la página 301

“PageHandler” en la página 194

Tareas relacionadas

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

Fuentes de información adicional acerca de EGL

La copia más reciente de este documento se encuentra en el siguiente sitio Web:

<http://www.ibm.com/developerworks/rational/library/egldoc.html>

Para obtener información detallada sobre cómo migrar el código fuente escrito en VisualAge Generator, consulte la publicación *VisualAge Generator to EGL Migration Guide* (archivo vagenmig.pdf), que se encuentra en el sitio Web mencionado anteriormente y en la sección del sistema de ayuda llamada *Instalar y migrar*.

Sin embargo, es aconsejable acceder al sitio Web mencionado anteriormente.

Si embargo, es recomendable acceder al sitio Web.

Conceptos relacionados

“Introducción a EGL” en la página 1

Visión general del lenguaje EGL

Proyectos, paquetes y archivos EGL

Un proyecto EGL incluye de cero a muchas carpetas fuente, cada una de las cuales incluye de cero a muchos paquetes, y cada uno de éstos incluye de cero a muchos archivos. Cada archivo contiene de cero a muchos componentes.

Proyecto EGL

Un proyecto EGL se caracteriza por un conjunto de propiedades, que se describen más adelante. En el contexto de un proyecto EGL, EGL realiza automáticamente la validación y resuelve las referencias a componentes cuando se realizan determinadas tareas; por ejemplo cuando se guarda un archivo EGL o archivo de construcción. Además, si trabaja con componentes de PageHandler (la salida de los cuales se utiliza para depurar aplicaciones Web en el entorno de prueba de WebSphere), EGL genera automáticamente la salida, pero sólo en este caso:

- Ha establecido el proceso de construcción automático después de seleccionar las siguientes opciones: **Ventana > Preferencias > Entorno de trabajo > Realizar construcción automáticamente en modificación de recurso**
- Ha establecido un descriptor de construcción por omisión como preferencia o propiedad

Un proyecto EGL se forma seleccionando **EGL** o **Web EGL** como tipo de proyecto cuando se crea un nuevo proyecto. Asigne las propiedades a medida que realiza los pasos de creación del proyecto. Para empezar a modificar las opciones después de completar estos pasos, pulse con el botón derecho del ratón en el nombre del proyecto y cuando se visualice un menú de contexto, pulse **Propiedades**.

Las propiedades de EGL son las siguientes:

Carpeta fuente EGL

Una o más carpetas de proyecto que son los directorios raíz de los paquetes del proyecto, cada uno de los cuales es un conjunto de subdirectorios. Una carpeta fuente es útil para mantener el fuente EGL separado de los archivos Java y para mantener los archivos fuente EGL fuera de los directorios de despliegue Web. Se recomienda especificar carpetas fuente EGL en todos los casos; pero si no se especifica una carpeta fuente, la única carpeta fuente es el directorio de proyectos.

El valor de esta propiedad se almacena en un archivo llamado `.eglp` del directorio de proyectos y se guarda en el depósito (si existe) que se utiliza para almacenar archivos EGL.

Cada uno de los asistentes de proyecto EGL crea una carpeta fuente llamada **EGLSource**.

Vía de acceso de construcción EGL

La lista de proyectos donde se busca algún componente que no se encuentra en el proyecto actual.

El valor de esta propiedad se almacena en un archivo llamado `.eglp` del directorio de proyectos y se guarda en el depósito (si existe) que se utiliza para almacenar archivos EGL.

En el siguiente ejemplo de un archivo .eglp, EGLSource es una carpeta fuente del proyecto actual y AnotherProject es un proyecto de la vía de acceso EGL:

```
<?xml version="1.0" encoding="UTF-8"?>
<eglp>
  <eglpentry kind="src" path="EGLSource"/>
  <eglpentry kind="src" path="\AnotherProject"/>
</eglp>
```

Las carpetas fuente de AnotherProject se determinan a partir del archivo .eglp de dicho proyecto.

Descriptores de construcción por omisión

Los descriptores de construcción que permiten generar rápidamente la salida, como se describe en *Generación en el entorno de trabajo*.

Paquete

Un paquete es una colección con nombre de componentes fuente relacionados. Cuando se crean componentes de construcción no se utiliza ningún paquete.

Por convenio, la exclusividad de los nombres de paquete se consigue haciendo que la parte inicial del nombre de paquete sea el nombre de dominio de Internet de la organización con el orden invertido. Por ejemplo, el nombre de dominio de IBM es ibm.com y los paquetes EGL empiezan por "com.ibm". Utilizando este convenio, se tiene la seguridad de que los nombres de los programas Web que desarrolla una organización no duplicarán los nombres de programas desarrollados por otra organización y los programas podrán instalarse en el mismo servidor sin que exista la posibilidad de se produzca un conflicto de nombres.

Las carpetas de un determinado paquete se identifican mediante el nombre de paquete, que es una secuencia de identificadores separados por puntos (.), como en el siguiente ejemplo:

```
com.mycom.mypack
```

Cada identificador corresponde a una subcarpeta de una carpeta fuente EGL. Por ejemplo, la estructura de directorios de com.mycom.mypack es \com\mycom\mypack, y los archivos fuente se almacenan en la carpeta más inferior; en este caso, en mypack. Si el espacio de trabajo es *c:\miEspacioTrabajo*, si el proyecto es *nuevo.proyecto* y si la carpeta fuente es EGLSource, la vía de acceso de este paquete es la siguiente:

```
c:\miEspacioTrabajo\nuevo.proyecto\EGLSource\com\mycom\mypack
```

Todos los componentes de un archivo EGL pertenecen al mismo paquete. La sentencia de paquete del archivo, si existe, especifica el nombre de dicho paquete. Si no se especifica una sentencia de paquete, los componentes se almacenan directamente en la carpeta fuente y se dice que están en el *paquete por omisión*. Se recomienda especificar siempre una sentencia de paquete ya que los archivos del paquete por omisión no pueden compartirse con los componentes de otros paquetes o proyectos.

Es posible que dos componentes con el mismo identificador no estén definidos en el mismo paquete. *Se recomienda no utilizar el mismo nombre de paquete en diferentes proyectos o en diferentes carpetas.*

El paquete para la salida Java generada es el mismo que paquete del archivo EGL.

Archivos EGL

Cada archivo EGL pertenece a una de las siguientes categorías:

Archivo fuente

Un archivo fuente EGL (extensión .egl) contiene componentes lógicos, de datos y de interfaz de usuario y está escrito en formato fuente EGL.

Cada uno de los *componentes generables* siguientes puede transformarse en una unidad compilable:

- DataTable
- FormGroup
- Manejador (la base de un manejador de informes)
- Biblioteca
- PageHandler
- Programa

Un archivo fuente EGL puede incluir de cero a muchos componentes no generables pero no puede incluir más de un componente generable. El componente generable (si existe) debe estar en el nivel superior del archivo y debe tener el mismo nombre que el archivo.

Archivo de construcción

Un archivo de construcción EGL (extensión .eglbld) puede contener cualquier número de componentes de construcción y está escrito en XML (Extensible Markup Language), en formato de archivo de construcción EGL. Puede revisar la DTD relacionada, que se encuentra en el siguiente directorio:

```
dirInstalación\egl\eclipse\plugins\  
com.ibm.etools.egl_versión
```

dirInstalación

El directorio de instalación del producto, como por ejemplo C:\Program Files\IBM\Rational\SPD\6.0. Si instaló y tuvo un producto de Rational Developer antes de instalar el producto que está utilizando ahora, deberá especificar el directorio utilizado en la instalación anterior.

versión

La versión instalada del conector; por ejemplo, 6.0.0

El nombre del archivo (como por ejemplo egl_wssd_6_0.dtd) empieza por las letras *egl* y un signo de subrayado. Los caracteres *wssd* hacen referencia a Rational Web Developer y Rational Application Developer, los caracteres *wsed* hacen referencia a Rational Application Developer para z/OS y los caracteres *wdsc* hacen referencia a Rational Application Developer para iSeries.

Después de añadir componentes a archivos, puede utilizar un depósito para mantener un historial de los cambios.

Recomendaciones

Esta sección ofrece recomendaciones para configurar los proyectos de desarrollo.

Para descriptores de construcción

Los proyectos de equipo deben asignar una persona como desarrollador de los descriptores de construcción. Las tareas que debe realizar esta persona son las siguientes:

- Crear los descriptores de construcción para los desarrolladores de código fuente
- Poner estos descriptores de construcción en un proyecto distinto de los proyectos de código fuente; y hacer que este proyecto distinto esté disponible en el depósito o que esté disponible mediante otros medios.
- Pedir a los desarrolladores de código fuente que establezcan la propiedad **descriptores de construcción por omisión** en los proyectos, de modo que la propiedad haga referencia a los descriptores de construcción correspondientes.
- Si un subconjunto pequeño de las opciones del descriptor de construcción (como por ejemplo para el ID de usuario y la contraseña) varía de un desarrollador de código fuente a otro, pida a cada desarrollador de código fuente que realice lo siguiente:
 - Codificar un descriptor de construcción personal que utilice la opción **nextBuildDescriptor** para que señale a un descriptor de construcción de grupo.
 - Pedir a los desarrolladores de código fuente que establezcan la propiedad **descriptores de construcción por omisión** en los archivos, carpetas o paquetes, de modo que la propiedad haga referencia al descriptor de construcción personal. La propiedad no se especifica a nivel de proyecto ya que la propiedad a nivel de proyecto está bajo el control del depósito, junto con otra información del proyecto.

Para obtener información adicional, consulte la sección *Componente descriptor de construcción*.

Para paquetes

Para paquetes, las recomendaciones son las siguientes:

- No utilice el mismo nombre de paquete en diferentes proyectos o directorios fuente
- No utilice el paquete por omisión

Asignación de componentes

En el caso de los componentes, muchas de las recomendaciones hacen referencia a una buena práctica y no a requisitos estrictos. Siga incluso las recomendaciones opcionales a menos que tenga un buen motivo para no hacerlo:

- Un *requisito* es que ponga los JSP en el mismo proyecto que los PageHandlers asociados.
- Si un componente no generable (como un componente de registro) sólo es utilizado por un programa, biblioteca o PageHandler, coloque el componente no generable en el mismo archivo que el componente que lo utiliza.
- Si se hace referencia a un componente desde diferentes archivos del mismo paquete, ponga dicho componente en un archivo distinto del paquete.
- Si un componente se comparte entre varios paquetes de un único proyecto, coloque dicho componente en un paquete distinto de dicho proyecto.
- Ponga el código para aplicaciones sin absolutamente ninguna relación entre ellas en proyectos distintos. El proyecto es la unidad para transferir el código entre la estructura de directorios local y el depósito. Diseñe una estructura de proyecto de modo que los desarrolladores puedan minimizar la cantidad de código que tienen que tener cargado en el sistema de desarrollo.

- Asigne nombres a los proyectos, paquetes y archivos de tal forma que se refleje el uso de los componentes que contienen.
- Si el proceso enfatiza la propiedad del código por parte de un desarrollador, no asigne componentes de diferentes propietarios al mismo archivo.
- Asigne componentes a paquetes con una descripción clara de la finalidad del paquete; y agrupe dichos componentes de acuerdo con relación de la proximidad entre ellos.

La siguiente distinción es importante:

- El hecho de mover un componente de un archivo a otro dentro del mismo paquete no obliga a cambiar las sentencias import en otros archivos.
- El hecho de mover un componente de un paquete a otro puede hacer necesario añadir o cambiar una sentencia import en cada archivo que haga referencia al componente que se ha movido.

Conceptos relacionados

“Componente descriptor de construcción” en la página 293

“Generación en el entorno de trabajo” en la página 330

“Referencias a componentes” en la página 21

“Import” en la página 33

“Introducción a EGL” en la página 1

“Componentes”

Consulta relacionada

“Formato de un archivo de construcción EGL” en la página 380

“Formato fuente EGL” en la página 491

“Sentencias EGL” en la página 88

Componentes

Un archivo EGL contiene un conjunto de *componentes* cada uno de los cuales es una unidad discreta y con nombre. Algunos componentes (como por ejemplo un programa) son *componentes generables*; cada uno de los cuales constituye la base de una unidad compilable. Un componente generable debe tener el mismo nombre que el archivo fuente EGL que contiene el componente.

Un archivo fuente EGL (extensión .egl) puede incluir cero o un componente generable y de cero a muchos otros componentes.

Los componentes se dividen en las categorías siguientes:

- Los *componentes lógicos* definen una secuencia de tiempo de ejecución que se escribe en el lenguaje de procedimiento EGL.
 - El componente no generable *function* es la unidad de lógica básica. El resto de especies de componentes lógicos pueden incluir funciones.
 - Puede definir dos tipos de *programas* que se diferencian por el tipo de interfaz. Cada uno es un componente generable:
 - Un *programa básico* evita la interacción con el usuario, o bien limita esa interacción a una especie determinada de interfaz basada en caracteres. La tecnología de interfaz en este caso funciona de la manera siguiente:
 - Visualiza la salida en un indicador de mandatos

- Permite que el usuario interactúe con el programa de forma inmediata, con cada pulsación de tecla definiendo potencialmente un suceso distinto para que lo maneje el programa.

Para obtener detalles sobre esta clase de interfaz, consulte la sección *Interfaz de usuario de consola*.

- Un *programa textUI* interactúa con el usuario de la forma siguiente:
 - Visualiza un conjunto de campos en un indicador de mandatos
 - Acepta la entrada de campo del usuario solo cuando el usuario pulsa una tecla de sumisión.

Puede definir cada tipo de programa para que sea un *programa principal*. Esa clase de programa se inicia de cualquiera de estas formas:

- Por el usuario
- Por una transferencia de programa que no sea una llamada
- Directamente por un proceso de sistema operativo

Además, puede declarar cualquiera de los dos tipos de programa para que sea un *programa llamado*, al que solamente puede invocarse mediante una llamada.

Para obtener información detallada sobre el despliegue en tiempo de ejecución de los programas principales y llamados, consulte *Configuraciones de tiempo de ejecución*.

- Un *PageHandler* es un componente generable que controla la interacción entre el usuario y una página Web.
- Un *manejador* del tipo *JasperReport* es un componente generable que contiene funciones personalizadas que se invocan en momentos distintos durante la ejecución de un archivo de diseño *JasperReports*. Los datos devueltos por cada función se incluyen en el informe de salida, que puede representarse en formato PDF, XML, texto o HTML.
- Una *biblioteca* también es un componente generable, un conjunto de funciones y variables compartidas que pueden ponerse a disposición de programas, *pageHandlers* y otras bibliotecas.
- Los *componentes de datos* definen las estructuras de datos que están disponibles en el programa.

Las clases siguientes de componentes de datos se utilizan como tipos en declaraciones de variable:

- Los componentes *DataItem* contienen información acerca de las clases de datos más elementales. Estos componentes son parecidos a las entradas de un diccionario de datos de todo el sistema; cada componente incluye detalles acerca del tamaño de los datos, el tipo, las reglas de formato, las reglas de validación de entrada y las sugerencias de visualización. Un componente *DataItem* se define una vez y puede utilizarlo como la base para cualquier número de variables primitivas o campos de registro.

El componente *DataItem* proporciona una forma adecuada de crear una variable a partir de un tipo primitivo. Por ejemplo, tenga en cuenta la definición siguiente de *myStringPart*, que es un componente *DataItem* de tipo serie:

```
DataItem
  MyStringPart String { validValues = ["abc", "xyz"] }
end
```

Cuando desarrolla una función, puede declarar una variable de tipo *MyStringPart*:

```
myString MyStringPart;
```

La declaración siguiente tiene el mismo efecto que la anterior:

```
myString STRING { validValues = ["abc", "xyz"] };
```

Tal como se muestra, el nombre de un componente `DataItem` es simplemente un alias para un tipo primitivo con valores de propiedad específicos.

- Los *componentes de registro* son una bases para datos complejos. Una variable cuyo tipos es un componente de registro incluye campos. Cada campo puede estar basado en cualquiera de los elementos siguientes:
 - Un tipo primitivo como por ejemplo `STRING`
 - Un componente `DataItem`
 - Un registro fijo (tal como se describe más adelante)
 - Otro componente de registro
 - Una matriz de cualquiera de las clases precedentes

Cada campo puede ser también un diccionario o un `ArrayDictionary` (tal como se describe posteriormente) o una matriz de diccionarios o `ArrayDictionaries`.

La variable basada en un componente de registro se llama registro y la longitud de los datos del registro puede variar en tiempo de ejecución.

Puede utilizar un componente de registro para crear variables para el proceso general o para acceder a una base de datos relacional.

- Los *componentes de registro fijo* son una base para los datos complejos que son de longitud fija. Una variable cuyo tipo es un componente de registro fijo incluye campos y cada campo puede tener cualquiera de los elementos siguientes como un tipo:
 - Un tipo primitivo como por ejemplo `CHAR`
 - Un componente `DataItem`

Cada campo puede estar subestructurado. Por ejemplo un campo que especifica un número de teléfono puede definirse de la forma siguiente:

```
10 phoneNumber CHAR(10);
20 areaCode CHAR(3);
20 localNumber CHAR(7);
```

Aunque puede utilizar componentes de registro fijo para cualquier clase de proceso, se utilizan principalmente para operaciones de E/S en archivos VSAM, colas de mensajes MQSeries y otros archivos secuenciales.

En cierta medida, EGL soporta componentes de registro fijos para permitir la compatibilidad con productos anteriores como por ejemplo VisualAge Generator. Aunque puede utilizar registros fijos para acceder a bases de datos relacionales o para proceso general, es recomendable que evite utilizar registros fijos para esos propósitos.

- Un *componente de diccionario* siempre está disponible; no se define. Una variable basada en un componente de diccionario puede incluir un conjunto de claves y los valores relacionados y puede añadir y eliminar entradas de clave y valor en tiempo de ejecución.
- Un *componente ArrayDictionary* siempre está disponible; no se define. Una variable basada en un componente `ArrayDictionary` permite acceder a una serie de matrices recuperando el mismo elemento numerado de cada matriz. Un conjunto de elementos recuperado de esta forma es en sí mismo un diccionario con cada nombre de matriz tratado como una clave emparejada con el valor en el elemento de matriz.

Un `ArrayDictionary` resulta especialmente útil en relación con la tecnología de visualización descrita en la sección *Interfaz de usuario de consola*.

El otro componente de datos es *DataTable*, que se trata como una variable en lugar de como tipo de variable. *DataTable* es un componente generable que pueden compartir varios programas. Contiene una serie de filas y columnas, incluye un valor primitivo en cada casilla y se trata como una variable que (en muchos casos) es global para la unidad de ejecución.

- Los *componentes UI (interfaz de usuario)* describen el diseño de los datos presentados al usuario en la pantalla de fonts fijos y los formularios de impresión. Los componentes de UI se utilizan en contextos distintos y son de los tipos siguientes:
 - Un *componente de registro* de subtipo *ConsoleForm* es una organización de datos que se presenta al usuario en el contexto de la tecnología consoleUI. Igual que otros componentes de registro, cada uno se utiliza como un tipo para una o varias variables pero, en este caso, cada variable se llama *formulario de consola* en lugar de registro. La tecnología ConsoleUI también incluye otros componentes que se definen automáticamente y que pueden utilizarse como base de las variables; para obtener detalles, consulte el apartado *Interfaz de usuario de consola*
 - Un *formulario* también es una organización de datos presentada al usuario. Una clase de formulario organiza los datos enviados a una pantalla en un programa textUI y otro organiza los datos enviados a una impresora en cualquier clase de programa.
Cada formulario incluye una estructura fija e interna como la de un registro fijo, pero un formulario no puede incluir una subestructura.
Un formulario sólo está a disposición de un programa, PageHandler o biblioteca si un FormGroup incluye o hace referencia al formulario, tal como se describe a continuación.
 - Un *componente FormGroup* es un conjunto de texto y formularios de impresión y es un componente generable. Un programa sólo puede incluir un formGroup para la mayoría de usos, junto con un formGroup para la salida relacionada con la ayuda. Puede incluirse el mismo formulario en múltiples FormGroups.
Los formularios de un FormGroup son globales para un programa, aunque el acceso debe especificarse en una sentencia de utilización específica de programa. Se hace referencia a los formularios como variables.
Las interfaces de usuario Web se crean con Page Designer, que crea un archivo JSP y lo asocia a un pageHandler EGL. El archivo JSP sustituye el cometido de un componente UI para las aplicaciones que interactúan con el usuario mediante la Web.
- Los *componentes de construcción* se definen en archivos de construcción EGL (extensión .eglbld) y definen diversas características de proceso:
 - Un *componente descriptor de construcción* controla el proceso de generación e indica qué otros componentes de control se leen durante dicho proceso.
 - Un *componente de opciones de enlace* proporciona detalles sobre cómo un programa generado se transfiere a otros programas. La información contenida en este componente se utiliza durante la generación, la prueba y la ejecución.
 - Un *componente de asociaciones de recursos* relaciona un registro EGL con la información necesaria para acceder a un archivo de una determinada plataforma destino; la información contenida en este componente se utiliza durante la generación, la prueba y la ejecución.

Un registro fijo, *DataTable* o formulario (ya sea texto o impresión) incluye una *estructura fija*. La estructura se compone de una serie de campos, cada uno de los

cuales tiene un tamaño y un tipo que se conocen durante la generación y, en el caso de DataTable o un registro fijo, el campo puede subestructurarse.

Conceptos relacionados

“ArrayDictionary” en la página 86

“Componente descriptor de construcción” en la página 293

“Compatibilidad con VisualAge Generator” en la página 439

“Interfaz de usuario de consola” en la página 177

“Componente dataItem” en la página 131

“Diccionario” en la página 82

“Proyectos, paquetes y archivos EGL” en la página 13

“Componentes de registro fijo” en la página 133

“Componente de función” en la página 140

“Import” en la página 33

“Introducción a EGL” en la página 1

“Componente de opciones de enlace” en la página 311

“Componente de programa” en la página 139

“Componentes de registro” en la página 132

“Referencias a componentes”

“Referencias a variables en EGL” en la página 58

“Asociaciones de recursos y tipos de archivo” en la página 304

“Configuraciones de tiempo de ejecución” en la página 9

“Estructura fija” en la página 26

“Typedef” en la página 27

“Soporte Web” en la página 187

Consulta relacionada

“Formato de un archivo de construcción EGL” en la página 380

“Editor EGL” en la página 483

“Formato fuente EGL” en la página 491

“Sentencias EGL” en la página 88

“Tipos primitivos” en la página 34

Referencias a componentes

Esta sección describe un conjunto de normas que determinan cómo EGL identifica el componente al que hace referencia un nombre. Estas normas son importantes en las siguientes situaciones:

- Una función invoca otra función
- Un componente que no es de función (por ejemplo, un componente dataItem) hace referencia a una función de validador
- Un componente actúa como una typedef (un modelo de formato) en la declaración de una variable o elemento de estructura
- Un componente hace referencia a otro en una declaración de uso
- Un componente de construcción hace referencia a otro

Un segundo conjunto de normas determina cómo EGL resuelve las referencias a variables. Para obtener información detallada, consulte la sección *Referencias a variables y constantes*.

Normas básicas de visibilidad

En el caso más simple, defina componentes uno tras otro en un único paquete, sin declarar un componente dentro de otro. La lista siguiente omite muchos detalles, pero muestra una serie de componentes que están en el mismo nivel jerárquico:

```
Función: Function01
Función: Function02
Función: Function03
Registro: Record01
```

Los componentes del mismo nivel están disponibles entre sí. Function01, por ejemplo, puede invocar una o las dos otras funciones; y Record01 puede utilizarse como una typedef para las variables de cada una de las tres funciones.

En la mayoría de casos, un componente *no puede* anidar otro componente. Las excepciones son las siguientes:

- Un programa, biblioteca o pageHandler puede anidar funciones, pero incluso entonces, la inclusión debe ser directa; una función no puede anidar otra función
- Un grupo de formularios puede anidar formularios

A continuación se ofrece un ejemplo con componentes anidados:

```
Programa: Program01
  Función: Function01
  Función: Function02
Función: Function03
Registro: Record01
```

Los componentes del nivel superior están disponibles en cualquier otro componente del paquete. Sin embargo, los componentes anidados (Function01 y Function02) sólo están disponibles en un subconjunto de componentes del paquete:

- Están disponibles entre sí.
- Están disponibles en el componente anidante y en las funciones que utiliza el componente anidante durante la ejecución. Si, por ejemplo, Function01 invoca Function03, Function03 puede invocar Function02 porque Function03 se utiliza en Program01.

Finalmente, si el código incluye formularios de texto o impresión, se necesita una declaración de uso para acceder al grupo de formularios que incluye dichos formularios. Una declaración de uso también es deseable cuando se accede a tablas de datos o a bibliotecas. Para obtener más información, consulte la sección *Declaración de uso*.

Normas adicionales de visibilidad

La mayoría de esfuerzos de desarrollo tienen componentes que se comparten en más de un paquete. Se aplican las siguientes normas:

- Cualquier componente del archivo puede hacer referencia a componentes de otros paquetes, a condición de que los componentes accedidos tengan las siguientes características:
 - Sean componentes de nivel superior
 - No estén declarados como *privados*
 - Estén en el mismo proyecto que el componente referenciador o estén en un proyecto listado en la vía de acceso de construcción EGL del proyecto referenciador

Puede proporcionar acceso de las siguientes maneras:

- Puede calificar el nombre de componente con el nombre de paquete, en cuyo caso no se necesita ninguna sentencia import en el archivo fuente. Si, por ejemplo, un nombre de paquete es *my.package*, y un nombre de componente es *myPart*, puede hacer referencia al componente de la siguiente manera:


```
my.package.myPart
```
- Puede utilizar sentencias import, que proporcionan las siguientes ventajas:
 - Las sentencias import hacen posible que no tenga que calificar los nombres de los componentes importados, a menos que utilice un nombre de paquete para evitar una referencia ambigua.
 - Las sentencias import proporcionan una forma de documentar los paquetes que se utilizan en el código fuente.

Resolución de nombres de componentes

Para resolver la referencia a un componente, EGL realiza una búsqueda que puede incluir uno o varios pasos. Se aplica lo siguiente *en cada paso*:

- La búsqueda finaliza satisfactoriamente si se encuentra un componente con un nombre exclusivo
- La búsqueda finaliza con un error si se encuentran dos componentes con el mismo nombre.

Son posibles las siguientes situaciones:

- La referencia al componente está calificada con un nombre de paquete; en este caso, la búsqueda siempre incluye un solo paso
- La referencia al componente no está calificada con un nombre de paquete y no es una invocación de función
- La referencia al componente no está calificada con un nombre de paquete y es una invocación de función

Las siguientes afirmaciones apenas son importantes, pero podrían aplicarse en una de las dos últimas situaciones:

- La propiedad **containerContextDependent** en la función referenciadora puede establecerse en *yes*. Establezca esta propiedad para ampliar el espacio de nombres utilizado para resolver referencias, como se describe en la sección *containerContextDependent*.
- Si una de las funciones está visible en el programa o PageHandler y tiene un nombre que es idéntico al nombre de una función de sistema EGL, se hace referencia a esta función y no a la función de sistema.

Resolución de nombres de componentes cuando el nombre de paquete está especificado: Como se ha indicado anteriormente, puede especificar el nombre de un paquete al hacer referencia a un componente, como en el ejemplo *my.package.myPart*. Se tiene en cuenta el proyecto actual, así como los proyectos listados en la vía de acceso de construcción EGL.

Si la referencia es de un componente que está dentro del mismo paquete, se aplica lo siguiente:

- El nombre de paquete es válido pero innecesario
- El nombre de paquete se resuelve aunque el componente esté declarado como privado

Resolución de nombres de componentes (que no sean una invocación de función) cuando el nombre de paquete no está especificado: Si un componente hace referencia a un componente que no es una función y no especifica un nombre de paquete, los pasos en el orden de búsqueda son los siguientes:

1. Busque los componentes anidados en el mismo contenedor que en el que está anidado el componente referenciador.
2. Busque los componentes que se han importado explícitamente en el archivo donde reside el componente referenciador. Se tiene en cuenta el proyecto actual, así como los proyectos listados en la vía de acceso de construcción EGL.

En este caso, cada sentencia *import* hace referencia explícitamente a un componente concreto de un determinado paquete. El componente especificado en esta *sentencia import de tipo explícito* actúa como una alteración temporal del componente con el mismo nombre en el paquete actual.

Si tiene paquetes con nombres idénticos en dos proyectos distintos, una determinada sentencia *import* de tipo explícito utiliza la vía de acceso de construcción EGL para realizar una búsqueda de primera coincidencia, deteniéndose cuando se encuentra el componente necesario. (El componente debe ser exclusivo de un paquete de un determinado proyecto). La presencia de un paquete con el mismo nombre en dos proyectos distintos no es un error, pero crea una situación de confusión y no se recomienda.

Se produce un error si tiene dos sentencias *import* de tipo explícito que hacen referencia al mismo componente.

3. Busque los componentes de nivel superior que están en el mismo paquete que el componente referenciador. Se tiene en cuenta el proyecto actual, así como los proyectos listados en la vía de acceso de construcción EGL. Se produce un error cuando se buscar dos componentes que tienen el mismo nombre.
4. Busque otros componentes importados. Se tiene en cuenta el proyecto actual, así como los proyectos listados en la vía de acceso de construcción EGL.

En este caso, cada sentencia *import* hace referencia a todos los componentes de un determinado paquete y se llama *sentencia import de tipo comodín*.

Si tiene paquetes con nombres idénticos en dos proyectos distintos, una determinada sentencia *import* de tipo comodín utiliza la vía de acceso de construcción EGL para realizar una búsqueda de primera coincidencia, deteniéndose cuando se encuentra el componente necesario. (El componente debe ser exclusivo de un paquete de un determinado proyecto).

Si más de una sentencia *import* de tipo comodín recupera el componente con el mismo nombre, se produce un error.

Invocación de función cuando el nombre de paquete no está especificado: Si un componente invoca una función y no especifica un nombre de paquete, los pasos en el orden de búsqueda son los siguientes:

1. Busque las funciones anidadas en el mismo contenedor que en el que está anidado el invocador.
2. Busque las funciones que residen en las bibliotecas especificadas en las declaraciones de uso del contenedor.
3. Continúe la búsqueda sólo con las funciones que se incluyen en el contenedor durante la generación. (Para incluir funciones que no son las que están anidadas en el mismo contenedor o que residen en una biblioteca, establezca la propiedad de contenedor **includeReferencedFunctions** en *sí*.)

La búsqueda de las funciones incluidas se realiza de la manera siguiente:

- a. Busque los componentes que se han importado explícitamente en el archivo donde reside el contenedor. Se tiene en cuenta el proyecto actual, así como los proyectos listados en la vía de acceso de construcción EGL.

En este caso, cada sentencia `import` hace referencia explícitamente a un componente concreto de un determinado paquete. El componente especificado en esta *sentencia import de tipo explícito* actúa como una alteración temporal del componente con el mismo nombre en el paquete actual.

Si tiene paquetes con nombres idénticos en dos proyectos distintos, una determinada sentencia `import` de tipo explícito utiliza la vía de acceso de construcción EGL para realizar una búsqueda de primera coincidencia, deteniéndose cuando se encuentra la función necesaria. (La función debe ser exclusiva de un paquete de un determinado proyecto). La presencia de un paquete con el mismo nombre en dos proyectos distintos no es un error, pero crea una situación de confusión y no se recomienda.

Se produce un error si tiene dos sentencias `import` de tipo explícito que hacen referencia al mismo componente.

- b. Busque las funciones de nivel superior del mismo paquete que el contenedor. Se tiene en cuenta el proyecto actual, así como los proyectos listados en la vía de acceso de construcción EGL. Se produce un error si la búsqueda encuentra dos componentes con el mismo nombre.
- c. Busque otros componentes importados. Se tiene en cuenta el proyecto actual, así como los proyectos listados en la vía de acceso de construcción EGL.

En este caso, cada sentencia `import` hace referencia a todos los componentes de un determinado paquete y se llama *sentencia import de tipo comodín*.

Si tiene paquetes con nombres idénticos en dos proyectos distintos, una determinada sentencia `import` de tipo comodín utiliza la vía de acceso de construcción EGL para realizar una búsqueda de primera coincidencia, deteniéndose cuando se encuentra el componente necesario. (El componente debe ser exclusivo de un paquete de un determinado proyecto).

Si más de una sentencia `import` de tipo comodín recupera el componente con el mismo nombre, se produce un error.

Invocación de programa

Cuando se invoca un programa en una sentencia `call` o `transfer`, la lista de argumentos del invocador debe coincidir con la lista de parámetros del programa invocado. Una discrepancia de argumento y parámetro produce un error.

Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13

“Import” en la página 33

“Introducción a EGL” en la página 1

“Componentes” en la página 17

“Referencias a variables en EGL” en la página 58

Consulta relacionada

“`containerContextDependent`” en la página 465

“Vía de acceso de construcción EGL y `eglpath`” en la página 477

“Editor EGL” en la página 483

“Formato fuente EGL” en la página 491

“Declaración `use`” en la página 954

Estructura fija

Una *estructura fija* establece el formato de un formulario de texto, un formulario de impresión, un dataTable o un componente de registro fijo y se compone de una serie de campos que cada uno describe como una ubicación de memoria elemental o un conjunto de ubicaciones de memoria, como en este ejemplo:

```
10 workAddress;  
  20 streetAddress1 CHAR(20);  
    30 Line1 CHAR(10);  
    30 Line2 CHAR(10);  
  20 streetAddress2 CHAR(20);  
    30 Line1 CHAR(10);  
    30 Line2 CHAR(10);  
  20 city CHAR(20);
```

Puede definir todos los campos directamente en la definición, como en el ejemplo anterior. O bien, puede indicar que la totalidad o un subconjunto de la estructura es equivalente a la estructura que está en otro componente de registro fijo; para obtener información detallada, consulte la sección *Typedef*.

El acceso a un campo se basa en un nombre de variable y luego en una serie de nombres de campo con una sintaxis por puntos. Si declara que el registro *myRecord* incluye la estructura que se muestra en el ejemplo anterior, cada uno de los siguientes identificadores hace referencia a un área de memoria:

```
myRecord.workAddress  
myRecord.workAddress.streetAddress1  
myRecord.workAddress.streetAddress1.Line1
```

Un *campo de estructura básico* no tiene campos de estructura subordinados y describe un área de memoria de una de las siguientes maneras:

- Mediante una especificación de longitud y tipo primitivo, como en el ejemplo anterior;
- Señalando a la declaración de un componente dataItem, como se describe en la sección *Typedef*.

Tal como se ha mostrado anteriormente, un campo de una estructura fija puede tener campos subordinados. Considere el siguiente ejemplo:

```
10 topMost;  
  20 next01 HEX(4);  
  20 next02 HEX(4);
```

Al definir un campo de estructura superior (como, por ejemplo, *topMost*), dispone de varias opciones:

- Si no asigna una longitud ni un tipo primitivo, el campo de estructura superior es de tipo CHAR y EGL calcula la longitud. Por ejemplo, el tipo primitivo de *topMost* es CHAR y la longitud es 4.
- Si asigna un tipo primitivo pero no asigna una longitud, EGL calcula la longitud basándose en las características de los elementos de estructura subordinados.
- Si asigna una longitud y un tipo primitivo, la longitud debe reflejar el espacio proporcionado para los campos de estructura subordinados; en caso contrario, se produce un error

Nota: El tipo primitivo de un campo de estructura determina el número de bytes de cada unidad de longitud; para obtener información detallada, consulte la sección *Tipos primitivos*.

Cada campo de estructura básico tiene una serie de propiedades, que pueden estar establecidas por omisión o pueden especificarse en el campo de estructura. (El campo de estructura puede hacer referencia a un componente `dataItem` que en sí mismo tiene propiedades). Para obtener información detallada, consulte la sección *Visión general de las propiedades y alteraciones temporales de EGL*.

Conceptos relacionados

“Componente `dataItem`” en la página 131
“Componentes de registro fijo” en la página 133
“Visión general de las propiedades de EGL” en la página 64
“Componentes” en la página 17
“Referencias a variables en EGL” en la página 58
“`typedef`”

Consulta relacionada

“Inicialización de datos” en la página 471
“Formato fuente EGL” en la página 491
“Tipos primitivos” en la página 34
“Propiedades de elementos SQL” en la página 67

Typedef

Una definición de tipo (`typedef`) es un componente que se utiliza como modelo de formato. Utilice el mecanismo `typedef` en los siguientes casos:

- Para identificar las características de una variable
- Para reutilizar las declaraciones de componente
- Para aplicar convenios de formato
- Para aclarar el significado de los datos

A menudo, las `typedef` identifican una agrupación abstracta. Por ejemplo, puede declarar un componente de registro llamado `address` y dividir dicha información en `streetAddress1`, `streetAddress2` y `city`. Si un registro de personal incluye los elementos de estructura `workAddress` y `homeAddress`, cada uno de estos elementos de estructura puede señalar al formato del componente de registro llamado `address`. Esta utilización de `typedef` asegura que los formatos de dirección sean iguales.

Dentro del conjunto de normas descritas en esta página, puede señalar al formato de un componente al declarar otro componente o al declarar una variable.

Cuando declare un componente, no es necesario que utilice un componente como una `typedef`, pero quizás desee hacerlo, como en los ejemplos que se muestran más adelante. Tampoco es necesario que utilice una `typedef` cuando declare una variable que tiene las características de un elemento de datos; en su lugar, puede especificar todas las características de la variable, sin hacer referencia a un componente.

Una `typedef` *siempre* está en vigor cuando se declara una variable que es más compleja que un elemento de datos. Por ejemplo, si declara una variable llamada `myRecord` y señala al formato de un componente llamado `myRecordPart`, EGL crea un modelo de la variable declarada en dicho componente. Si, en cambio, señala al formato de un componente llamado `myRecordPart02`, la variable se llama `myRecord` pero tiene todas las características del componente llamado `myRecordPart02`.

La tabla y las secciones siguientes proporcionan detalles sobre las typedef en diferentes contextos.

Entrada que señala a una typedef	Tipo de componente al que la typedef puede hacer referencia
parámetro de función u otra variable de función	un componente de registro o componente dataItem
parámetro de programa	componente dataItem, componente form, componente record
variable de programa (no parámetro)	componente dataItem, componente record
elemento de estructura	componente dataItem, componente record

Componente dataItem como una typedef

Puede utilizar un componente de elemento de datos (dataItem) como una typedef en las siguientes situaciones:

- Al declarar una variable o parámetro
- Al declarar un elemento de estructura, que es una subunidad de un componente de registro, componente de formulario o componente dataTable

Se aplican las siguientes normas:

- Si un elemento de estructura es el padre de otros elementos de estructura que están listados en la misma declaración, el elemento de estructura sólo puede señalar al formato de un componente dataItem, como en el siguiente ejemplo:

```
DataItem myPart CHAR(20) end

Record myRecordPart type basicRecord
  10 mySI myPart; // myPart actúa como una typedef
  20 a CHAR(10);
  20 b CHAR(10);
end
```

El componente de registro anterior es equivalente a la siguiente declaración:

```
Record myRecordPart type basicRecord
  10 mySI CHAR(20);
  20 a CHAR(10);
  20 b CHAR(10);
end
```

- No se puede utilizar un componente de dataItem como una typedef y además especificar la longitud o tipo primitivo de la entidad que señala a la typedef, como en el siguiente ejemplo:

```
DataItem myPart HEX(20) end

// NO válido porque mySI tiene un tipo primitivo
// y señala al formato de un componente (a myPart, en este caso)
Record myRecordPart type basicRecord
  10 mySI CHAR(20) myPart;
end
```

- Una declaración de variable que no hace referencia a un componente de registro señala al formato de un componente de dataItem o bien tiene características primitivas. (Un parámetro de programa también puede hacer referencia a un componente de formulario). Sin embargo, un componente dataItem no puede señalar al formato de otro componente dataItem ni a ningún otro componente.
- Un componente de registro SQL sólo puede utilizar los siguientes tipos de componentes como typedef:
 - Otro componente de registro SQL

- Un componente dataItem

Componente de registro como una typedef

Puede utilizar un componente de registro como una typedef en las siguientes situaciones:

- Al declarar un elemento de estructura
- Al declarar una variable (incluido un parámetro), en cuyo caso la variable refleja la typedef de las siguientes maneras:
 - Formato
 - Tipo de registro (por ejemplo, indexedRecord o serialRecord)
 - Valores de propiedad (por ejemplo, el valor de la propiedad `file`)

Cuando declare un elemento de estructura que señala al formato de otro componente, especifique si la typedef añade un nivel de jerarquía, como se ilustra más adelante.

Se aplican las siguientes normas:

- Un componente de registro puede ser una typedef cuando se utiliza un elemento de estructura para facilitar la reutilización:

```
Record address type basicRecord
  10 streetAddress1 CHAR(30);
  10 streetAddress2 CHAR(30);
  10 city CHAR(20);
end

Record record1 type serialRecord
{
  fileName = "myFile"
}
  10 person CHAR(30);
  10 homeAddress address;
end
```

El segundo componente de registro es equivalente a la siguiente declaración:

```
Record record1 type serialRecord
{ fileName = "myFile" }
  10 person CHAR(30);
  10 homeAddress;
  20 streetAddress1 CHAR(30);
  20 streetAddress2 CHAR(30);
  20 city CHAR(20);
end
```

Si un elemento de estructura utiliza la sintaxis anterior para señalar al formato de un componente de estructura, EGL añade un nivel jerárquico al componente de estructura que incluye el elemento de estructura. Por este motivo, la estructura interna del ejemplo anterior tiene una jerarquía de elementos de estructura, donde *person* está en un nivel distinto de *streetAddress1*.

- En algunos casos, es preferible una organización plana de la estructura; y un registro SQL que está en un objeto de E/S para el acceso a la base de datos relacional *debe* tener una organización de este tipo.
 - En el ejemplo anterior, si sustituye la palabra **embed** para el nombre de elemento de estructura de un componente de registro (en este caso, *homeAddress*) y a continuación de esta palabra añade el nombre del componente de registro que actúa como una typedef (en este caso, *address*), las declaraciones del componente tienen el siguiente aspecto:

```
Record address type basicRecord
  10 streetAddress1 CHAR(30);
  10 streetAddress2 CHAR(30);
```

```

    10 city CHAR(20);
end

Record record1 type serialRecord
{
  fileName = "myFile"
}
  10 person CHAR(30);
  10 embed address;
end

```

La estructura interna del componente de registro ahora es plana:

```

Record record1 type serialRecord
{
  fileName = "myFile"
}
  10 person CHAR(30);
  10 streetAddress1 CHAR(30);
  10 streetAddress2 CHAR(30);
  10 city CHAR(20);
end

```

El único motivo por el que se utiliza la palabra **embed** en lugar de un nombre de elemento de estructura es para no tener que añadir un nivel de jerarquía. Un elemento de estructura identificado mediante la palabra **embed** tiene las siguientes restricciones:

- Puede señalar al formato de un componente de registro, pero no a un componente dataItem
 - No puede especificar una matriz ni incluir una especificación de tipo primitivo
- A continuación, considere el caso en que un componente de registro es una typedef cuando declara estructuras idénticas en dos registros:

```

Record common type serialRecord
{
  fileName = "mySerialFile"
}
  10 a BIN(10);
  10 b CHAR(10);
end

Record recordA type indexedRecord
{
  fileName = "myFile",
  keyItem = "a"
}
  embed common; // acepta la estructura de common,
                // no las propiedades
end

Record recordB type relativeRecord
{
  fileName = "myOtherFile",
  keyItem = "a"
}
  embed common;
end

```

Los dos últimos componentes de registro son equivalentes a las siguientes declaraciones:

```

Record recordA type indexedRecord
{
  fileName = "myFile",
  keyItem = "a"
}
  10 a BIN(10);

```

```

        10 b CHAR(10);
    end

    Record recordB type relativeRecord
    {
        fileName = "myOtherFile",
        keyItem = "a"
    }
    10 a BIN(10);
    10 b CHAR(10);
end

```

- Puede utilizar un componente de registro varias veces como una typedef al declarar una serie de elementos de estructura. Esta reutilización tiene sentido, por ejemplo, si declara un componente de registro de personal que incluye una dirección particular y una dirección de trabajo. Un registro básico podría proporcionar el mismo formato en dos ubicaciones de la estructura:

```

Record address type basicRecord
    10 streetAddress1 CHAR(30);
    10 streetAddress2 CHAR(30);
    10 city CHAR(20);
end

Record record1 type serialRecord
{
    fileName = "myFile"
}
    10 person CHAR(30);
    10 homeAddress address;
    10 workAddress address;
end

```

El componente de registro es equivalente a la siguiente declaración:

```

Record record1 type serialRecord
{
    fileName = "myFile"
}
    10 person CHAR(30);
    10 homeAddress;
        20 streetAddress1 CHAR(30);
        20 streetAddress2 CHAR(30);
        20 city CHAR(20);
    10 workAddress;
        20 streetAddress1 CHAR(30);
        20 streetAddress2 CHAR(30);
        20 city CHAR(20);
end

```

- No se puede utilizar un componente de registro como una typedef y además especificar la longitud o tipo primitivo de la entidad que señala a la typedef, como en el siguiente ejemplo:

```

Record myTypedef type basicRecord
    10 next01 HEX(20);
    10 next02 HEX(20);
end

// no válido porque myFirst tiene un
// tipo primitivo y señala al formato de un componente
Record myStruct02 type serialRecord
{
    fileName = "myFile"
}
    10 myFirst HEX(40) myTypedef;
end

```

Sin embargo, considere el siguiente caso:

```
Record myTypedef type basicRecord
  10 next01 HEX(20);
  10 next02 HEX(20);
end
```

```
Record myStruct02 type basicRecord
  10 myFirst myTypedef;
end
```

La segunda estructura es equivalente a la siguiente declaración:

```
Record myStruct02 type basicRecord
  10 myFirst;
  20 next01 HEX(20);
  20 next02 HEX(20);
end
```

El tipo primitivo de cualquier elemento de estructura que tenga elementos de estructura subordinados es CHAR por omisión, y la longitud de dicho elemento de estructura es el número de bytes representado por los elementos de estructura subordinados, independientemente de los tipos primitivos de dichos elementos de estructura. Para obtener información detallada, consulte la sección Estructura.

- Se aplican las siguientes restricciones en relación con los registros SQL:
 - Si un componente de registro SQL utiliza otro componente de registro SQL como una typedef, cada elemento que proporciona la typedef incluye un prefijo de cuatro bytes. Sin embargo, si un registro no SQL utiliza un componente de registro SQL como una typedef, no se incluye ningún prefijo. Para obtener información general, consulte la sección *Componentes internos de un registro SQL*.
 - Un componente de registro SQL sólo puede utilizar los siguientes tipos de componentes como typedef:
 - Otro componente de registro SQL
 - Un componente dataItem
- Finalmente, ni una estructura ni un elemento de estructura puede ser una typedef

Formulario como una typedef

Sólo puede utilizar un componente de formulario como una typedef cuando declare un parámetro de programa.

Conceptos relacionados

“Componente dataItem” en la página 131
 “Componente de formulario” en la página 154
 “Introducción a EGL” en la página 1
 “Componentes de registro” en la página 132
 “Estructura fija” en la página 26

Tareas relacionadas

“Crear un componente de programa de EGL” en la página 138

Consulta relacionada

“Sentencias EGL” en la página 88
 “Diseño interno de los registros SQL” en la página 747

Import

Una sentencia `import` identifica un conjunto de componentes que están en un paquete especificado (para archivos fuente EGL) o en un conjunto especificado de archivos (para archivos de construcción EGL). El archivo que contiene la sentencia `import` puede hacer referencia a los componentes importados como si estuvieran en el mismo paquete que el archivo.

Información general

Si un componente público reside en un paquete que no es el paquete actual pero no está identificado en una sentencia `import`, el código debe calificar el nombre de componente (por ejemplo, `myPart`) con el nombre de paquete (por ejemplo, `my.pkg`), como en el siguiente ejemplo:

```
my.pkg.myPart
```

Sin embargo, si el componente está identificado en una sentencia `import`, el código puede eliminar el nombre del paquete. En este caso, el nombre de componente no calificado (como, por ejemplo, `myPart`) es suficiente.

Si desea ver una descripción de las circunstancias en las que se utilizan las sentencias `import` para resolver un nombre de componente, consulte la sección *Referencias a componentes*.

Formato de la sentencia import

La sintaxis de la sentencia `import` en un archivo EGL es la siguiente:

```
import packageName.partSelection;
```

nombrePaquete

Identifica el nombre de un paquete en el que se debe buscar. Tiene que ser un nombre completo.

selecciónComponente

Es un nombre de componente o un asterisco (*). El asterisco indica que todos los componentes del paquete están seleccionados.

Una sentencia `import` de un archivo de construcción identifica otros archivos de construcción a cuyos componentes se puede hacer referencia mediante los componentes del archivo importador. Las sentencias `import` siguen el código <EGL> del archivo de construcción y cada sentencia tiene la siguiente sintaxis:

```
<import file=víaAccesoArchivo.eglbld>
```

víaAccesoArchivo

Identifica la vía de acceso y el nombre del archivo que se debe importar. Si se especifica una vía de acceso, se aplica lo siguiente:

- La vía de acceso del archivo está en alguno de los directorios fuente del mismo proyecto o en cualquier otro proyecto que está en la vía de acceso EGL
- Cada calificador se separa del siguiente con una barra inclinada (/)

Puede especificar un asterisco (*) como nombre de archivo o como último carácter del nombre de archivo. Si se utiliza el asterisco, EGL importa todos los archivos `.eglbld` con las siguientes características:

- Están en la vía de acceso de archivo especificada.

- Tienen nombres que empiezan con caracteres que preceden al asterisco. (Si el asterisco no tiene caracteres que lo precedan, se seleccionan todos los archivos de construcción existentes en la vía de acceso del directorio).

La extensión de archivo .eglbl es opcional.

Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13

“Introducción a EGL” en la página 1

“Componentes” en la página 17

“Referencias a componentes” en la página 21

Tareas relacionadas

“Editar un vía de acceso de construcción de EGL” en la página 320

Tipos primitivos

Cada tipo primitivo EGL caracteriza un área de memoria. Estas son las tres clases de tipos primitivos: de carácter, numéricos y de fecha y hora.

- Las tipos de carácter son los siguientes:
 - *CHAR* hace referencia a caracteres de un solo byte.
 - *DBCHAR* hace referencia a caracteres de doble byte. *dbchar* sustituye a *DBCS*, que era un tipo primitivo en EGL V5.
 - *MBCHAR* hace referencia a caracteres multibyte, que son una combinación de caracteres de un solo byte y de doble byte. *mbchar* sustituye a *MIX*, que era un tipo primitivo en EGL V5.
 - *STRING* hace referencia a un campo de longitud variable en la que los caracteres de doble byte se ajustan a los estándares de codificación UTF-16 desarrollados por Unicode Consortium.
 - *UNICODE* hace referencia a un campo fijo en el que los caracteres de doble byte se ajustan a los estándares de codificación UTF-16 desarrollados por Unicode Consortium.
 - *HEX* hace referencia a caracteres hexadecimales.
- Los tipos de fecha y hora son los siguientes:
 - *DATE* hace referencia a una fecha de calendario específica que tiene una longitud fija de ocho dígitos de un solo byte.
 - *INTERVAL* hace referencia a un intervalo de tiempo cuya longitud va de dos a veintisiete dígitos de un solo byte.
 - *TIME* hace referencia a una instancia temporal con una longitud de seis dígitos de un solo byte.
 - *TIMESTAMP* hace referencia a la hora actual y tiene una longitud que va de dos a veintisiete dígitos de un solo byte.
- Los tipos de objeto grandes son los siguientes:
 - *BLOB* hace referencia a un objeto grande con una longitud que va de un byte a dos gigabytes.
 - *CLOB* hace referencia a un objeto de caracteres grande con una longitud que va de un byte a dos gigabytes.
- Las tipos numéricos son los siguientes:
 - *BIGINT* hace referencia a un área de 8 bytes que almacena un entero de 18 dígitos como máximo. Este tipo es equivalente al tipo *BIN*, longitud 8, sin posiciones decimales.

- *BIN* hace referencia a un número binario.
- *DECIMAL* hace referencia a caracteres decimales empaquetados cuyo signo está representado por una C hexadecimal (para un número positivo) o por una D hexadecimal (para un número negativo) en la mitad derecha del byte situado más a la derecha. *DECIMAL* sustituye a *PACK* que era un tipo primitivo en la versión 5.0 de EGL.
- *FLOAT* hace referencia a un área de 8 bytes que almacena números de coma flotante de precisión doble con 16 dígitos significativos como máximo.
- *INT* hace referencia a un área de 4 bytes que almacena un entero de 9 dígitos como máximo. Este tipo es equivalente al tipo *BIN*, longitud 4, sin posiciones decimales.
- *MONEY* hace referencia a las cantidades de divisa, que se almacenan como valores *DECIMAL*.
- *NUM* hace referencia a caracteres numéricos cuyo signo está representado por un valor hexadecimal específico de signo en la mitad izquierda del byte situado más a la derecha. En ASCII, ese valor es 3 (para un número positivo) y 7 (para un número negativo); en EBCDIC, ese valor es F (para un número positivo) y D (para un número negativo).
- *NUMC* hace referencia a caracteres numéricos cuyo signo está representado por un valor hexadecimal específico de signo en la mitad izquierda del byte situado más a la derecha. En ASCII, ese valor es 3 (para un número positivo) y 7 (para un número negativo); en EBCDIC, ese valor es F (para un número positivo) y C (para un número negativo).
- *PACF* hace referencia a caracteres decimales empaquetados cuyo signo está representado por una F hexadecimal (para un número positivo) o por una D hexadecimal (para un número negativo) en la mitad derecha del byte situado más a la derecha.
- *SMALLFLOAT* hace referencia a un área de 4 bytes que almacena un número de coma flotante de precisión simple, como 8 dígitos significativos como máximo.
- *SMALLINT* hace referencia a un área de 2 bytes que almacena un entero de 4 dígitos como máximo. Este tipo es equivalente al tipo *BIN*, longitud 2, sin posiciones decimales.

La representación interna de un campo de cualquiera de los tipos numéricos de coma fija es la misma que una representación de entero, aunque especifique una coma decimal. Por ejemplo, la representación de 12.34 es la misma que la de 1234. De la misma forma, no se almacenan símbolos de divisa con campos de tipo *MONEY*.

Al interactuar con DB2 (directamente o a través de JDBC) , el número máximo de dígitos de un número de coma fija es 31.

Una variable de tipo *ANY* recibe el tipo de valor asignado a esa variable, tal como se describe en el tipo *ANY*.

Durante la declaración, se especifica el tipo primitivo que caracteriza cada uno de estos valores:

- El valor devuelto por una función
- El valor de un campo, que es un área de memoria a la que se hace referencia por nombre y contiene un solo valor

Otras entidades también tienen un tipo primitivo:

- Una variable de sistema tiene un tipo primitivo (generalmente NUM) que es específico del campo
- Un literal de carácter pertenece a uno de los siguientes tipos:
 - CHAR si el literal incluye sólo caracteres de un solo byte
 - DBCHAR si el literal incluye sólo caracteres de doble byte del juego de caracteres de doble byte
 - MBCHAR si el literal incluye una combinación de caracteres de un solo byte y de doble byte
- Los literales de caracteres de tipo UNICODE no están soportados.

Cada tipo primitivo se describe en una página independiente, y encontrará detalles adicionales en las páginas que describen las asignaciones, las expresiones lógicas, las invocaciones de función y la sentencia call.

Las secciones que siguen describen estos temas:

- Tipos primitivos durante la declaración
- Eficacia relativa de los diversos tipos numéricos

Tipos primitivos durante la declaración

Considere las siguientes declaraciones:

```
DataItem
  myItem CHAR(4)
end
Record mySerialRecordPart type serialRecord
{
  fileName="myFile"
}
10 name CHAR(20);
10 address;
  20 street01 CHAR(20);
  20 street02 CHAR(20);
end
```

Como se muestra, debe especificar un tipo primitivo al declarar estas entidades:

- Una variable primitiva
- Un campo de estructura que no tenga subestructuras

Puede especificar el tipo primitivo de un campo de estructura subestructurado como *address*. Si no especifica el tipo primitivo de un campo de estructura de este tipo, pero hace referencia al campo de estructura en el código, el producto hace las siguientes suposiciones:

- Se presupone que el tipo primitivo es CHAR, aunque los campos de estructura subordinados sean de un tipo diferente
- Se presupone que la longitud será el número de bytes de los campos de estructura subordinados

Eficacia relativa de los diversos tipos numéricos

EGL da soporte a los tipos DECIMAL, NUM, NUMC y PACF a fin de que el usuario pueda trabajar más fácilmente con archivos y bases de datos utilizados por aplicaciones de legado. Es aconsejable utilizar campos de tipo BIN en desarrollos nuevos o utilizar un tipo entero equivalente (BIGINT, INT o SMALLINT); los cálculos son más eficaces con los campos de estos tipos. Obtendrá la mayor eficacia utilizando campos de tipo BIN, longitud 2 y sin posiciones decimales (el equivalente del tipo SMALLINT).

En los cálculos, asignaciones y comparaciones, los campos de tipo NUM sin posiciones decimales son más eficaces que los campos de tipo NUM con posiciones decimales.

Los cálculos con campos de tipo DECIMAL, NUM, NUMC y PACF son igualmente eficaces.

Conceptos relacionados

“Componente dataItem” en la página 131
“Componentes de registro” en la página 132
“Referencias a variables en EGL” en la página 58
“Estructura fija” en la página 26

Consulta relacionada

“ANY”
“Asignaciones” en la página 374
“BIN y los tipos enteros (integer)” en la página 50
“call” en la página 563
“CHAR” en la página 38
“DATE” en la página 41
“DBCHAR” en la página 38
“DECIMAL” en la página 50
“Manejo de excepciones” en la página 94
“FLOAT” en la página 51
“Invocaciones de función” en la página 518
“HEX” en la página 39
“INTERVAL” en la página 42
“Expresiones lógicas” en la página 497
“MBCHAR” en la página 39
“MONEY” en la página 51
“NUM” en la página 51
“NUMC” en la página 52
“Expresiones numéricas” en la página 504
“Operadores y precedencia” en la página 673
“PACF” en la página 52
“SMALLFLOAT” en la página 53
“Propiedades de elementos SQL” en la página 67
“STRING” en la página 40
“Expresiones de texto” en la página 505
“TIME” en la página 43
“TIMESTAMP” en la página 44
“UNICODE” en la página 40

ANY

Una variable de tipo ANY recibe el tipo de valor asignado a esa variable. El valor puede ser de un tipo primitivo como por ejemplo INT o puede ser una variable basada en un componente de datos utilizado como un tipo. El valor no puede ser un formulario ni dataTable.

Considere el siguiente ejemplo:

```
myInt INT = 1;  
myString STRING = "EGL";  
  
myAny01, myAny02 any;  
  
// myAny01 recibe el valor 1 y el tipo INT
```

```

myAny01 = myInt;

// myAny02 recibe el valor "EGL" y el tipo STRING
myAny02 = myString;

// La sentencia siguiente
// NO ES VÁLIDA porque una variable de tipo INT
// se asigna a una variable de tipo STRING
myAny02 = myAny01;

```

Las acciones que combinan tipos de forma no válida solo se detectan en tiempo de ejecución e implican la terminación del programa. Esas acciones incluyen la asignación de un valor a un campo de tipo incompatible, pasar un valor de argumento a un parámetro de tipo incompatible o combinar valores incompatibles dentro de una expresión.

El tipo de un literal numérico está implícito en el valor del literal:

- Una serie entrecomillada es de tipo STRING
- Un entero de 4 dígitos o menos es de tipo SMALLINT
- Un entero de 5 a 8 dígitos es de tipo INT
- Un entero de 9 a 18 dígitos es de tipo BIGINT
- Un número que incluye una coma decimal es de tipo NUM

Cuando hace referencia a una variable de tipo ANY, el acceso siempre es dinámico. No puede incluir un campo de tipo ANY en una estructura fija (dataTable, formulario de impresión, formulario de texto o registro fijo).

Consulta relacionada

“Tipos primitivos” en la página 34

Tipos de caracteres

CHAR

Un elemento de tipo CHAR se interpreta como una serie de caracteres de un solo byte. La longitud refleja tanto el número de caracteres como el número de bytes y va de 1 a 32767.

Las plataformas de estación de trabajo como Windows 2000 utilizan el juego de caracteres ASCII; las plataformas de sistema principal como z/OS UNIX System Services utilizan el juego de caracteres EBCDIC. Generalmente, las diferencias del orden de clasificación provocan que las comparaciones de tipo mayor que y menor que produzcan resultados diferentes en los dos tipos de entornos.

Consulta relacionada

“Tipos primitivos” en la página 34

DBCHAR

Un elemento de tipo DBCHAR se interpreta como una serie de caracteres de doble byte. La longitud refleja el número de caracteres y va de 1 a 16383. Para determinar el número de bytes, doble el valor de longitud.

Las plataformas de estación de trabajo como Windows 2000 utilizan el juego de caracteres ASCII; las plataformas de sistema principal como z/OS UNIX System Services utilizan el juego de caracteres EBCDIC. Generalmente, las diferencias del orden de clasificación provocan que las comparaciones de tipo mayor que y menor que produzcan resultados diferentes en los dos tipos de entornos.

Los datos DBCS son ideográficos, necesarios para visualizar los idiomas chino, japonés y coreano, por ejemplo. La visualización de este tipo de datos requiere un dispositivo de terminal con capacidad para DBCS.

Consulta relacionada

“Tipos primitivos” en la página 34

HEX

Un elemento de tipo HEX se interpreta como una serie de dígitos hexadecimales (0-9, a-f y A-F), que se tratan como caracteres. La longitud refleja el número de dígitos y va de 1 a 65534. Para determinar el número de bytes, divídala por 2.

Para un elemento de longitud 4, las representaciones de bits internas de valores de ejemplo son las siguientes:

```
// valor hexadecimal 04 D2
00000100 11010010

// valor hexadecimal FB 2E
11111011 00101110
```

El uso principal de un elemento de tipo HEX es acceder a un campo de archivo o base de datos cuyo tipo de datos no coincide con otro tipo primitivo EGL.

Puede asignar un valor hexadecimal utilizando un literal de tipo CHAR que incluya sólo caracteres en el rango de dígitos hexadecimales, como en los siguientes ejemplos:

```
myHex01 = "ab02";

myHex02 = "123E";
```

Puede incluir un elemento hexadecimal como operando de una expresión lógica, como en los siguientes ejemplos:

```
if (myHex01 = "aBCd")
  myFunction01();
else
  if (myHex > myHex02)
    myFunction02();
  end
end
```

Puede incluir un elemento hexadecimal en una expresión aritmética.

Consulta relacionada

“Tipos primitivos” en la página 34

MBCHAR

Un elemento de tipo MBCHAR se interpreta como una combinación de caracteres de un solo byte y de doble byte. La longitud refleja el número de caracteres de un solo byte que el elemento puede contener y también el número de bytes. La longitud va de 1 a 32767.

Las plataformas de estación de trabajo como Windows 2000 utilizan el juego de caracteres ASCII; las plataformas de sistema principal como z/OS UNIX System Services utilizan el juego de caracteres EBCDIC. Generalmente, las diferencias del orden de clasificación provocan que las comparaciones de tipo mayor que y menor que produzcan resultados diferentes en los dos tipos de entornos.

En un entorno de sistema principal, debe incluir espacio para caracteres de desplazamiento a teclado ideográfico y a teclado estándar si es posible que el elemento contenga caracteres de doble byte.

- Un carácter de desplazamiento a teclado ideográfico de un solo byte (valor hexadecimal 0E) indica el principio de una serie de caracteres de doble byte
- Un carácter de desplazamiento a teclado estándar (valor hexadecimal 0F) indica el final de esa serie

Los caracteres de carácter de desplazamiento a teclado ideográfico y a teclado estándar se suprimen durante una conversión de datos de EBCDIC a ASCII y se insertan durante una conversión de datos de ASCII a EBCDIC. Si se convierte un registro de longitud variable, y si el fin de registro actual (según lo indicado por la longitud de registro) se encuentra dentro de un elemento de estructura de tipo MBCHAR, la longitud de registro se ajusta para reflejar la inserción o supresión de los caracteres de desplazamiento a teclado ideográfico y a teclado estándar.

Los datos de carácter de doble byte son ideográficos, necesarios para visualizar los idiomas chino, japonés y coreano, por ejemplo. La visualización de este tipo de datos requiere un dispositivo de terminal con capacidad para juegos de caracteres de doble byte.

Consulta relacionada

“Tipos primitivos” en la página 34

STRING

El tipo primitivo STRING se componen de caracteres UNICODE de doble byte.

Puede almacenar el valor del campo en un archivo o en una base de datos. Si el código interactúa con DB2 UDB, debe asegurarse de que la página de códigos de los datos de tipo GRAPHIC sea UNICODE y de que la columna que almacena el valor de elemento de datos sea del tipo de datos SQL GRAPHIC o VARGRAPHIC.

Para obtener detalles acerca de Unicode, consulte el sitio Web de Unicode Consortium (www.unicode.org).

Consulta relacionada

“Tipos primitivos” en la página 34

UNICODE

El tipo primitivo UNICODE ofrece una forma de procesar y almacenar texto que puede encontrarse en cualquiera de diversos idiomas humanos; sin embargo, el texto debe haberse suministrado desde fuera del código. Los literales de tipo UNICODE no están soportados.

En un elemento de tipo UNICODE, se cumplen las siguientes afirmaciones:

- La longitud refleja el número de caracteres y va de 1 a 16383. El número de bytes reservado para un elemento de este tipo es el doble del valor especificado para la longitud.
- El elemento sólo puede asignarse a o compararse con otro elemento de tipo UNICODE.
- Todas las comparaciones comparan los valores de bit de acuerdo con el orden de los caracteres en el estándar de codificación UTF-16.
- Si es necesario, EGL rellena el elemento con blancos Unicode.

- Las funciones de serie del sistema tratan el elemento como una serie de bytes individuales, incluidos los blancos Unicode añadidos, si los hay. Las longitudes que especifique en dichas funciones deben expresarse en términos de bytes, en lugar de en términos de caracteres.
- Puede almacenar el valor de un elemento en un archivo o en una base de datos. Si el código interactúa con DB2 UDB, debe asegurarse de que la página de códigos de los datos de tipo GRAPHIC sea UNICODE y de que la columna que almacena el valor de elemento de datos sea del tipo de datos SQL GRAPHIC o VARGRAPHIC.

Para obtener detalles acerca de Unicode, consulte el sitio Web de Unicode Consortium (www.unicode.org).

Consulta relacionada

“Tipos primitivos” en la página 34

Tipos de fecha y hora

DATE

Un elemento de tipo DATE es una serie de dígitos numéricos de un solo byte que reflejan una fecha de calendario específica.

El formato de tipo DATE es *aaaaMMdd*:

aaaa

Cuatro dígitos que representan un año. El rango va de 0000 a 9999.

MM

Dos dígitos que representan un mes. El rango va de 01 a 12.

dd Dos dígitos que representan un día. El rango va de 01 a 31 y se produce un error si el código asigna una fecha no válida como por ejemplo 20050230.

La representación hexadecimal interna de un valor de ejemplo es como sigue si el elemento está en un entorno de sistema principal que utiliza EBCDIC:

```
// Marzo 15, 2005
F2 F0 F0 F5 F0 F3 F1 F5
```

La representación hexadecimal interna de un valor de ejemplo es la siguiente si el elemento está en un entorno de estación de trabajo como por ejemplo Windows 2000 que utiliza ASCII:

```
// Marzo 15, 2005
32 30 30 35 30 33 31 35
```

Un elemento de tipo DATE puede recibir y proporcionar datos de y a una base de datos relacional.

Consulta relacionada

“Biblioteca DateTimeLib de EGL” en la página 791

“Expresiones de fecha y hora” en la página 495

“Tipos primitivos” en la página 34

“Especificadores de fecha, hora e indicación de la hora” en la página 45

INTERVAL

Un elemento de tipo INTERVAL es una serie de uno a veintiún dígitos numéricos de un solo byte que reflejan una intervalo que es la diferencia numérica entre dos puntos en el tiempo. El significado de cada dígito viene determinado por la máscara que especifica cuando declara el elemento.

Un intervalo puede ser positivo (como al restar 1980 de 2005) o negativo (como al restar 2005 de 1980) y al principio del elemento hay un byte extra que no se refleja en la máscara. Si un elemento de tipo INTERVAL está en un registro, debe contar con el byte extra al calcular la longitud del registro, así como la longitud del elemento superior, si lo hay.

Puede especificar una máscara que esté en cualquiera de estos dos formatos:

- Tramo de mes, que puede incluir años y meses
- Segundo tramo, que puede incluir días, horas, minutos, segundos y fracciones de segundos

En cualquier caso, cada carácter de la máscara representa un dígito. En el formato de tramo de mes, por ejemplo, el conjunto de *a* indica cuantos años hay en elemento. Si solo necesita tres dígitos para representar el número de años, especifique *aaa* en la máscara. Si necesita el número máximo de dígitos (nueve) para representar el número de años, especifique *aaaaaaaaa*.

En una más cara dada, el primer carácter puede utilizarse hasta nueve veces (a menos que se indique lo contrario) pero el número de cada especie de caracteres subsiguiente está más restringido.

Para una máscara con el formato de tramo de mes, están disponibles los caracteres siguientes, indicados por orden:

- a* De cero a nueve dígitos que representan el número de años del intervalo.
- M* De cero a nueve dígitos que representan el número de meses del intervalo. Si *M* no es el primer carácter de la máscara, solo se permiten dos dígitos como máximo.

La máscara por omisión es *aaaaMM*.

Para una máscara que tenga el formato de tramo de segundos, están disponibles los caracteres siguientes, indicados por orden:

- d* De cero a nueve dígitos que representan el número de días del intervalo.
- H* De cero a nueve dígitos que representan el número de horas del intervalo. Si *H* no es el primer carácter de la máscara, solo se permiten dos dígitos como máximo.
- m* De cero a nueve dígitos que representan el número de minutos del intervalo. Si *m* no es el primer carácter de la máscara, solo se permiten dos dígitos como máximo.
- s* De cero a nueve dígitos que representan el número de segundos del intervalo. Si *s* no es el primer carácter de la máscara, solo se permiten dos dígitos como máximo.
- f* De cero a seis dígitos que representan cada uno una fracción de segundos; el

primero representa décimas, el segundo representa centésimas, etc. Incluso cuando *f* es el primer carácter de la máscara, solo se permiten seis dígitos como máximo.

Aunque puede tener cero caracteres de una clase dada al principio o al final de una máscara, no puede saltar caracteres intermedios. Las máscaras válidas incluyen estas:

```
aaaaaaMM
aaaaaa
MM

ddHHmssffffff
HHmssff
mss
HHmm
```

Sin embargo, las máscaras siguientes no son válidas porque faltan los caracteres intermedios:

```
// NO válido
ddmssffffff
HHsfff
```

La representación hexadecimal de un valor de ejemplo es la siguiente si la máscara por omisión (*aaaaMM*) está en vigor y si el elemento está en un entorno de sistema principal que utiliza EBCDIC:

```
// 100 años, 2 meses; 4E significa que el valor es positivo
4E F0 F1 F0 F0 F0 F2

// 100 años, 2 meses; 60 significa que el valor es negativo
60 F0 F1 F0 F0 F0 F2
```

La representación hexadecimal interna de un valor de ejemplo es la siguiente si la máscara por omisión (*aaaaMM*) está en vigor y si el elemento está en un entorno de estación de trabajo como Windows 2000, que utiliza ASCII:

```
// 100 años, 2 meses; 2B significa que el valor es positivo
2B 30 31 30 30 30 32

// 100 años, 2 meses; 2D significa que el valor es negativo
2D 30 31 30 30 30 32
```

Un elemento de tipo INTERVAL es de tipos fuertes, por lo que no puede comparar un elemento de este tipo con un elemento de ningún otro tipo ni puede asignar un elemento de cualquier otro tipo a o desde un elemento de este tipo.

Finalmente, un elemento de tipo INTERVAL no puede recibir datos de o proporcionar datos a una base de datos relacional.

Consulta relacionada

“Biblioteca DateTimeLib de EGL” en la página 791

“Expresiones de fecha y hora” en la página 495

“Tipos primitivos” en la página 34

“Especificadores de fecha, hora e indicación de la hora” en la página 45

TIME

Un elemento de tipo TIME es una serie de 6 dígitos numéricos de un solo byte que reflejan un momento específico.

El formato de tipo TIME es *HHmmss*:

HH

Dos dígitos que representan la hora. El rango va de 00 a 24.

mm

Dos dígitos que representan el minuto dentro de la hora. El rango va de 00 a 59.

ss Dos dígitos que representan el segundo dentro del minuto. El rango va de 00 a 59.

La representación hexadecimal interna de un valor de ejemplo es como sigue si el elemento está en un entorno de sistema principal que utiliza EBCDIC:

```
// 8:40:20  
F0 F8 F4 F0 F2 F0
```

La representación hexadecimal interna de un valor de ejemplo es la siguiente si el elemento está en un entorno de estación de trabajo como por ejemplo Windows 2000 que utiliza ASCII:

```
// 8:40:20  
30 38 34 30 32 30
```

Un elemento de tipo TIME puede recibir y proporcionar datos de y a una base de datos relacional.

Consulta relacionada

“Biblioteca DateTimeLib de EGL” en la página 791

“Expresiones de fecha y hora” en la página 495

“Tipos primitivos” en la página 34

“Especificadores de fecha, hora e indicación de la hora” en la página 45

TIMESTAMP

Un elemento de tipo TIMESTAMP es una serie de uno a veinte dígitos numéricos de un solo byte que reflejan un momento específico. El significado de cada dígito viene determinado por la máscara que especifica cuando declara el elemento.

Los caracteres siguientes están disponibles, por orden, cuando especifica la máscara:

aaaa

Cuatro dígitos que representan el año. El rango va de 0000 a 9999.

MM

Dos dígitos que representan el mes. El rango va de 01 a 12.

dd

Dos dígitos que representan el día. El rango va de 01 a 31.

HH

Dos dígitos que representan la hora. El rango va de 00 a 23.

mm

Dos dígitos que representan el minuto. El rango va de 00 a 59.

ss

Dos dígitos que representan el segundo. El rango va de 00 a 59.

f

De cero a seis dígitos que representan cada uno una fracción de segundos; el primero representa décimas, el segundo representa centésimas, etc.

La máscara por omisión es *aaaaMMddHHmmss*.

Cuando interactúa con DB2 (directamente o a través de JDBC) , debe especificar cada componente desde el año (*aaaa*) hasta los segundos (*ss*). En otros contextos, se cumple lo siguiente:

- Puede tener cero caracteres de una clase dada al principio o al final de una máscara, pero no puede saltar caracteres intermedios.
- Las máscaras válidas incluyen estas:

```
aaaaMMddHHmmss
aaaa
MMss
```

- Las máscaras siguientes no son válidas porque faltan los caracteres intermedios:

```
// NO válido
ddMMssffffff
HHssff
```

La representación hexadecimal interna de un valor de ejemplo es la siguiente si la máscara por omisión (*aaaaMMddHHmmss*) está en vigor y si el elemento está en un entorno de sistema principal que utiliza EBCDIC:

```
// 8:05:10 a 12 enero 2005
F2 F0 F0 F5 F0 F1 F1 F2 F0 F8 F0 F5 F1 F0
```

La representación hexadecimal interna de un valor de ejemplo es la siguiente si la máscara por omisión (*aaaaMMddHHmmss*) está en vigor y si el elemento está en un entorno de estación de trabajo como Windows 2000, que utiliza ASCII:

```
// 8:05:10 a 12 enero 2005
32 30 30 35 30 31 31 32 30 38 30 35 31 30
```

Un elemento de tipo `TIMESTAMP` puede compararse con (o asignarse a o desde) un elemento de tipo `TIMESTAMP` o un elemento de tipo `DATE`, `TIME`, `NUM` o `CHAR`. Sin embargo, se produce un error en tiempo de desarrollo si asigna un valor que no es válido. A continuación se ofrece un ejemplo:

```
// NO válido porque el 30 de febrero no es una fecha válida
myTS timestamp("aaaamdd");
myTS = "20050230";
```

Si faltan los caracteres iniciales de una máscara completa (por ejemplo, si la máscara es "dd"), EGL supone que los caracteres de nivel superior ("aaaaMM", en este caso) representen el momento actual, de acuerdo con el reloj del sistema. Las sentencias siguientes originan un error de tiempo de ejecución en febrero:

```
// NO válido porque el 30 de febrero no es una fecha
myTS timestamp("dd");
myTS = "30";
```

Finalmente, un elemento de tipo `TIMESTAMP` puede recibir datos de o proporcionar datos a una base de datos relacional.

Consulta relacionada

"Asignaciones" en la página 374

"Especificadores de fecha, hora e indicación de la hora"

"Expresiones de fecha y hora" en la página 495

"Biblioteca `DateTimeLib` de EGL" en la página 791

"Expresiones lógicas" en la página 497

"Tipos primitivos" en la página 34

Especificadores de fecha, hora e indicación de la hora

Los formatos de fechas, horas e indicaciones de la hora se especifican mediante un patrón de letras, representando cada letra un componente de la fecha o la hora.

Estos caracteres son sensibles a las mayúsculas y minúsculas y todas las letras desde la a hasta la z y desde la A hasta la Z se analizan como componentes de la fecha o la hora.

Para visualizar letras en la fecha, la hora o en la indicación de la hora sin que el texto se analice como un componente de fecha u hora, incluya esa letra o letras entre apóstrofes. Para visualizar un apóstrofe en la fecha, la hora o la indicación de la hora, utilice dos apóstrofes.

En la tabla siguiente se listan las letras y sus valores en un patrón de fecha, hora o indicación de la hora.

Letra	Componente de fecha u hora	Tipo	Ejemplos
G	Designador de era	Texto	AD
a	Año	Año	1996; 96
M	Mes del año	Mes	Julio; Jul; 07
w	Semana del año	Número	27
W	Semana del mes	Número	2
D	Día del año	Número	189
d	Día del mes	Número	10
F	Día de la semana en el mes	Número	2
E	Día de la semana	Texto	Martes; Mar
p	Marcador AM/PM	Texto	PM
H	Hora del día (0-23)	Número	0
k	Hora del día (1-24)	Número	24
K	Hora en AM/PM (0-11)	Número	0
h	Hora en AM/PM (1-12)	Número	12
m	Minuto de la hora	Número	30
s	Segundo del minuto	Número	55
S	Milisegundo	Número	978
z	Huso horario	Huso horario general	Hora estándar del pacífico; PST; GMT-08:00
Z	Huso horario	Huso horario RFC 822	-800
C	Siglo	Siglo	20; 21

El número de cada letra utilizado consecutivamente en el patrón determina cómo se interpreta y analiza ese grupo de letras. La interpretación depende del tipo de letra. Además, la interpretación depende de si el patrón se utiliza para formatear o analizar. La lista siguiente describe los tipos de letras y cómo afecta a la interpretación la diferencia en el número de esas letras.

Texto Para el formato, si el número de letras es menor que 4, se utiliza la forma completa. De lo contrario, se utiliza una abreviatura, si la hay. Al analizar, se aceptan ambas formas, independientemente del número de letras de patrón.

Número

Para el formato, el número de letras del patrón representa el número mínimo de dígitos. Se se añaden ceros a los números mas cortos para que alcancen la longitud designada. Para el análisis, el número de letras del patrón se ignora a menos que sea necesario separar dos campos adyacentes.

Año Para el formato, si el número de letras del patrón es 2, el año se trunca a 2 dígitos. De lo contrario, se interpreta como el tipo de número.

Para el análisis, si el número de letras del patrón no es 2, el año se interpreta literalmente, independientemente del número de dígitos. Por ejemplo, el patrón MM/dd/aaaa asignado al valor 01/11/12 se analiza como Enero 11, 12 A.D. El mismo patrón asignado al valor 01/02/3 o 01/02/0003 se analiza como Enero 2, 3 A.D. De la misma forma, el mismo patrón asignado al valor 01/02/-3 se analiza como Enero 2, 4 A.C.

Para el análisis, si el patrón es aa, el analizador determina el año completo relativo al año actual. El analizador presupone que el año de dos dígitos está dentro de los 80 años anteriores o de los 20 posteriores al momento del proceso. Por ejemplo, si el año actual es 2004, el patrón MM/dd/aa asignado al valor 01/11/12 se analiza como Enero 11, 2012, mientras que el mismo patrón asignado al valor 05/04/64 se analiza como Mayo 4, 1964.

Mes Si el número de letras de patrón es 3 o más, el mes se interpreta como tipo de texto. De lo contrario, se interpreta como el tipo de número.

Huso horario general

Los husos horarios generales se interpretan como el tipo de texto si tienen nombres. Para los husos horarios que representan un valor de desplazamiento GMT, se utiliza la sintaxis siguiente:

`GMToffsetTimeZone = GMT Signo Horas : Minutos`

Signo O bien +, o bien -

Horas Un número de uno o dos dígitos entre 0 y 23. El formato es independiente del entorno local y debe tomarse del bloque Basic Latin del estándar Unicode.

Minutos

Un número de dos dígitos entre 00 y 59. El formato es independiente del entorno local y debe tomarse del bloque Basic Latin del estándar Unicode.

Para el análisis, también se aceptan los husos horarios RFC 822.

Huso horario RFC 822

Se utiliza el formato de huso horario de 4 dígitos de RFC 822

`RFC822TimeZone = Signo horasDosDígitos : Minutos`

HorasDosDígitos debe ser un número de dos dígitos entre 00 y 23. Las otras definiciones son iguales que el tipo de huso horario General.

Para el análisis, también se aceptan los husos horarios General.

Siglo Si visualiza como un tipo de número que toma la modalidad de año completo por 100.

La tabla siguiente muestra algunos ejemplos de patrones de fecha y hora interpretados en el entorno local de EE.UU.

Patrón de fecha y hora	Resultado
aaaa.MM.dd G 'a las' HH:mm:ss z	2001.07.04 AD a las 12:08:56 PDT
EEE, MMM d, 'aa	Mie, Jul 4, '01
h:mm p	12:08 PM
hh 'horas' a, zzzz	12 horas PM, Hora del pacífico
K:mm p, z	0:08 PM, PDT
aaaaa.MMMMM.dd GGG hh:mm ppp	02001.July.04 AD 12:08 PM
EEE, d MMM aaaa HH:mm:ss Z	Mie, 4 Jul 2001 12:08:56 -0700
aaMMdHHmssZ	010704120856-0700

Tipos LOB

CLOB

Un elemento de tipo CLOB representa un objeto de caracteres grande con una longitud que va de un byte a dos gigabytes.

En un elemento de tipo CLOB, se cumplen las siguientes afirmaciones:

- Solo puede declararse como elemento individual y no está soportado en BasicRecords.
- Puede pasarse a una función local y a llamadas de programa. Los parámetros de objetos grandes y los argumentos correspondientes deben declararse como objetos grandes del mismo tipo.
- Sólo puede asignarse a otra variable Clob.
- Puede moverse a otra variable Clob, lo que es equivalente a asignarse a una variable Clob.
- Puede crear una variable de referencia de BLOB.
- Utiliza SQLlocator (CLOB); es decir, CLOB contiene un puntero lógico a los datos CLOB SQL en lugar de a los datos en sí mismos.
- Cuando se utiliza con SQLRecord,
 - CLOB representa un objeto de caracteres grande como una columna en la base de datos.
 - CLOB es válido mientras dura la conversión en la que se creó.
- No puede pasarse a llamadas a programas remotos ni a programas no EGL.
- No es posible que se le haga referencia como un operando en sentencias de asignación ni en expresiones.

Puede utilizar las funciones siguientes con CLOB:

- attachClobToFile
- freeClob
- getClobLen
- getStrFromClob
- getSubStrFromClob
- loadClobFromFile
- setClobFromString

- setClobFromStringAtPosition
- truncateClob
- updateClobToFile

Consulta relacionada

"BLOB"

"Biblioteca LobLib de EGL" en la página 831

"attachClobToFile()" en la página 833

"freeClob()" en la página 834

"getClobLen()" en la página 834

"getStrFromClob()" en la página 835

"getSubStrFromClob()" en la página 835

"loadClobFromFile()" en la página 836

"setClobFromString()" en la página 836

"setClobFromStringAtPosition()" en la página 837

"truncateClob()" en la página 837

"updateClobToFile()" en la página 838

"Tipos primitivos" en la página 34

BLOB

Un elemento de tipo BLOB representa un objeto binario grande con una longitud que va de un byte a dos gigabytes.

En un elemento de tipo BLOB, se cumplen las siguientes afirmaciones:

- Solo puede declararse como elemento individual y no está soportado en BasicRecords.
- Puede pasarse a una función local y a llamadas de programa. Los parámetros de objetos grandes y los argumentos correspondientes deben declararse como objetos grandes del mismo tipo.
- Sólo puede asignarse a otra variable Blob.
- Puede moverse a otra variable Blob, lo que es equivalente a asignarse a una variable Blob.
- Puede crear una variable de referencia de BLOB.
- Utiliza SQLlocator (BLOB); es decir, BLOB contiene un puntero lógico a los datos BLOB SQL en lugar de a los datos en sí mismos.
- Cuando se utiliza con SQLRecord,
 - BLOB representa un objeto binario grande como una columna en la base de datos.
 - BLOB es válido mientras dura la conversión en la que se creó.
- No puede pasarse a llamadas a programas remotos ni a programas no EGL.
- No es posible que se le haga referencia como un operando en sentencias de asignación ni en expresiones.

Puede utilizar las funciones siguientes con BLOB:

- attachBlobToFile
- freeBlob
- getBlobLen
- loadBlobFromFile
- truncateBlob
- updateBlobToFile

Consulta relacionada

"CLOB" en la página 48
"Biblioteca LobLib de EGL" en la página 831
"attachBlobToFile()" en la página 832
"freeBlob()" en la página 833
"getBlobLen()" en la página 834
"loadBlobFromFile()" en la página 835
"truncateBlob()" en la página 837
"updateClobToFile()" en la página 838
"Tipos primitivos" en la página 34

Tipos numéricos

BIN y los tipos enteros (integer)

Un elemento de tipo BIN se interpreta como valor binario. La longitud puede ser 4, 9 o 18 y refleja el número de dígitos positivos en formato decimal, incluidas las posiciones decimales. Por ejemplo, el valor -12.34 cabe en un elemento de longitud 4. Un número de 4 dígitos requiere 2 bytes; un número de 9 dígitos requiere 4 bytes; y un número de 18 dígitos requiere 8 bytes.

Para un elemento de longitud 4, las representaciones de bits internas de valores de ejemplo son las siguientes:

```
// para decimal 1234, el valor hexadecimal es 04 D2:  
00000100 11010010  
  
// para decimal -1234, el valor es el complemento de 2 (FB 2E):  
11111011 00101110
```

Es aconsejable utilizar elementos de tipo BIN en lugar de otros tipos numéricos siempre que sea posible; por ejemplo, para operandos o resultados aritméticos, para subíndices de matriz y para elementos de clave de registros relativos.

Los siguientes tipos son equivalentes al tipo BIN:

- BIGINT tiene la longitud 18, sin posiciones decimales
- INT tiene la longitud 9, sin posiciones decimales
- SMALLINT tiene la longitud 4, sin posiciones decimales

Consulta relacionada

"Tipos primitivos" en la página 34

DECIMAL

Un elemento de tipo DECIMAL es un valor numérico en el que cada medio byte es un carácter hexadecimal, y el signo está representado por una C hexadecimal (para un número positivo) o por una D hexadecimal (para un número negativo) en la mitad derecha del byte situado más a la derecha.

La longitud refleja el número de dígitos y va de 1 a 32.

Para determinar el número de bytes, añada 2 al valor de longitud, divida la suma por 2 y trunque las fracciones en el resultado.

Para un elemento de longitud 4, las representaciones hexadecimales internas de valores de ejemplo son las siguientes:

```
// para decimal 123  
00 12 3C
```

```
// para decimal -123  
00 12 3D
```

```
// para decimal 1234  
01 23 4C
```

```
// para decimal -1234  
01 23 4D
```

Un valor negativo que se lee de un archivo o base de datos en un campo de tipo DECIMAL puede tener una B hexadecimal en lugar de una D; EGL acepta el valor, pero convierte la B en D.

El formato de una columna DB2 UDB de tipo DECIMAL es equivalente al formato de una variable de lenguaje principal de tipo DECIMAL.

Consulta relacionada

“Tipos primitivos” en la página 34

FLOAT

Un elemento de tipo FLOAT se interpreta como un valor binario para números de coma flotante de doble precisión con un máximo de 16 dígitos significativos. La longitud se fija en 8 bytes. En los programas Java generados por EGL, el valor va de 4,9e-324 a 1,7976931348623157e308.

FLOAT corresponde a cada una de estas definiciones:

- El tipo de datos FLOAT en un sistema de gestión de base de datos relacional
- El tipo de datos **double** en C, C++ o Java

Para los valores de coma flotante, el formato de conversión entre los formatos Java y COBOL de sistema principal está soportado por DB2 pero no lo está en las llamadas a programas de sistema principal.

Consulta relacionada

“Tipos primitivos” en la página 34

MONEY

Un elemento de tipo MONEY es un valor numérico equivalente en muchos aspectos a un elemento de tipo DECIMAL. En el caso de MONEY, la longitud predeterminada es 16, el valor predeterminado para el número de posiciones decimales es 2, la longitud mínima es 2 y se visualiza un símbolo de moneda en los campos de salida. MONEY corresponde al tipo de datos MONEY de Informix 4GL de IBM.

El formato se basa en la variable defaultMoneyFormat.

Consulta relacionada

“DECIMAL” en la página 50

“Propiedades de formato” en la página 66

“Tipos primitivos” en la página 34

NUM

Un elemento de tipo NUM es un valor numérico en el que cada byte es un dígito en formato de caracteres, y el signo está representado por un valor hexadecimal

específico de signo en la mitad izquierda del byte situado más a la derecha. La longitud refleja tanto el número de dígitos como el número de bytes. La longitud va de 1 a 32.

Para un elemento de longitud 4, las representaciones hexadecimales internas de valores de ejemplo son las siguientes si el elemento está en un entorno de sistema principal que utiliza EBCDIC: :

```
// para decimal 1234
F1 F2 F3 F4

// para decimal -1234
F1 F2 F3 D4
```

Las representaciones hexadecimales internas de valores de ejemplo son las siguientes si el elemento está en un entorno de estación de trabajo como Windows 2000, que utiliza ASCII:

```
// para decimal 1234
31 32 33 34

// para decimal -1234
31 32 33 74
```

Consulta relacionada

“Tipos primitivos” en la página 34

NUMC

Un campo de tipo NUMC es un valor numérico en el que cada byte es un dígito en formato de caracteres, y el signo está representado por un valor hexadecimal específico de signo en la mitad izquierda del byte situado más a la derecha. La longitud refleja tanto el número de dígitos como el número de bytes y va de 1 a 18.

Para un campo de longitud 4, las representaciones hexadecimales internas de valores de ejemplo son las siguientes si el campo está en un entorno de sistema principal que utiliza EBCDIC: :

```
// para decimal 1234
F1 F2 F3 C4

// para decimal -1234
F1 F2 F3 D4
```

Las representaciones hexadecimales internas de valores de ejemplo son las siguientes si el campo está en un entorno de estación de trabajo como Windows 2000, que utiliza ASCII:

```
// para decimal 1234
31 32 33 34

// para decimal -1234
31 32 33 74
```

Consulta relacionada

“Tipos primitivos” en la página 34

PACF

Un campo de tipo PACF es un valor numérico en el que cada medio byte es un carácter hexadecimal, y el signo está representado por una F hexadecimal (para un número positivo) o por una D hexadecimal (para un número negativo) en la mitad derecha del byte situado más a la derecha. La longitud refleja el número de dígitos

y va de 1 a 18. Para determinar el número de bytes, añada 2 al valor de longitud, divida la suma por 2 y trunque las fracciones en el resultado.

Para un campo de longitud 4, las representaciones hexadecimales internas de valores de ejemplo son las siguientes:

```
// para decimal 123
00 12 3F

// para decimal -123
00 12 3D

// para decimal 1234
01 23 4F

// para decimal -1234
01 23 4D
```

Un valor negativo que se lee de un archivo o base de datos en un campo de tipo PACF puede tener una B hexadecimal en lugar de una D; EGL acepta el valor, pero convierte la B en D.

Consulta relacionada

“Tipos primitivos” en la página 34

SMALLFLOAT

Un elemento de tipo SMALLFLOAT se interpreta como un valor binario para números de coma flotante de precisión simple con un máximo de 8 dígitos significativos. La longitud se fija en 4 bytes de almacenamiento de memoria.

En los programas Java generados por EGL, el valor va de 3,40282347e+38 a 1,40239846e-45.

SMALLFLOAT corresponde a cada una de estas definiciones:

- El tipo de datos SMALLFLOAT en un sistema de gestión de base de datos relacional
- El tipo de datos **float** en C, C++ o Java

Para los valores de coma flotante, el formato de conversión entre los formatos Java y COBOL de sistema principal está soportado por DB2 pero no lo está en las llamadas a programas de sistema principal.

Consulta relacionada

“Tipos primitivos” en la página 34

Declarar variables y constantes en EGL

Puede declarar una variable de las siguientes maneras:

- Puede basar variable en uno o varios tipos primitivos, como en este ejemplo:
`myItem CHAR(10);`
- Puede basar una variable en un componente dataItem, un componente de registro o un componente de registro fijo, como en este ejemplo:
`myRecord myRecordPart;`
- Puede basar una variable en la configuración específica de un diccionario o de un arrayDictionary, como en este ejemplo:

```

myVariable Dictionary
{
    empnum=0005,
    lastName="Twain",
    firstName="Mark",
    birthday="021460"
};

```

- Un programa u otro componente generable puede acceder a los campos de una dataTable, que se trata como una variable global del programa o de la unidad de ejecución. Puede utilizar una sintaxis más sencilla para acceder a estos campos si dataTable se lista en una de las declaraciones de utilización del programa.
- Un programa puede acceder a los campos de un formulario de impresión o de texto que se trata como una variable global del programa. El programa debe incluir el formGroup relacionado en una declaración de utilización.
- Un programa u otro componente de lógica generable puede acceder a las variables de biblioteca que se declaran fuera de cualquier función de biblioteca. Estas variables son globales para la unidad de ejecución. Puede utilizar una sintaxis más sencilla para acceder a estos campos si la biblioteca se lista en una de las declaraciones de utilización del programa.

Puede declarar una constante especificando el símbolo CONST seguido por el nombre de constante, el tipo, el signo igual y el valor; el valor especificado no puede cambiarse en tiempo de ejecución. A continuación se ofrecen algunos ejemplos:

```

const myString String = "Great software!";
const myArray BIN[] = [36, 49, 64];
const myArray02 BIN[][] = [[1,2,3],[5,6,7]];

```

Una constante no puede estar en un registro ni en otra estructura compleja.

Finalmente, para declarar varias variables o constantes en una sola sentencia, separe un identificador del siguiente mediante una coma, como se hace en estos ejemplos:

```

const myString01, myString02 STRING = "INITIAL";
myItem01, myItem02, myItem03 CHAR(5);
myRecord01, myRecord02 myRecordPart;

```

Conceptos relacionados

“Referencias a componentes” en la página 21

“Componentes” en la página 17

“Typedef” en la página 27

Consulta relacionada

“Tipos primitivos” en la página 34

“Declaración use” en la página 954

Acceso dinámico y estático

EGL resuelve una referencia de variable mediante el acceso estático o dinámico:

- Cuando el *acceso dinámico* está en vigor, el nombre del campo y el tipo se conocen solo en tiempo de ejecución. El código determina el nombre de un valor en el código o de la entrada en tiempo de ejecución.

El acceso dinámico está en vigor cuando el código hace referencia a cualquiera de los elementos siguientes:

- Una variable cuyo tipo primitivo es ANY.
- Un campo de valor de un diccionario; el campo es de tipo ANY.

- Un campo de un registro, cuando la cadena de relaciones que llevan a ese campo (de registro a campo y a subcampo) es tal que una referencia anterior utiliza el acceso dinámico.
- Un campo al que se hace referencia mediante la sintaxis de corchetes de EGL. En este caso, el nombre del campo no sigue necesariamente las reglas para identificadores, pero puede ser una palabra reservada EGL o puede incluir espacios y otros caracteres que no serían válidos de otra forma.
Para conocer más detalles, consulte la sección *Sintaxis de corchetes para el acceso dinámico*.
- Cuando el *acceso estático* está en vigor, el nombre del campo y el tipo se conocen durante la generación y el nombre siempre se ajusta al convenio de denominación de los identificadores de EGL. El nombre no se utiliza en tiempo de ejecución.
El acceso estático está en vigor cuando el código hace referencia a cualquiera de los elementos siguientes:
 - Una variable que está fuera de cualquier contenedor y cuyo tipo es distinto de ANY
 - Un campo de un registro fijo
 - Un campo de un registro no fijo cuando la cadena de relaciones que llevan a ese campo (de variable a campo y a subcampo) es tal que cada referencia utiliza el acceso estático

Considere un ejemplo en el que los valores de un diccionario incluyen un registro fijo y un registro no fijo:

```
// un componente de registro fijo
Record myFixedRecordPart type=serialRecord
{
  fileName = "myFile"
}
10 ID INT;
10 Job CHAR(10);
end

// un componente de registro (no fijo)
Record myDynamicRecordPart type=basicRecord
ID INT;
Job CHAR(10);
end

Program myProgram

dynamicPerson myDynamicRecordPart;
myFlexID INT;

fixedPerson myFixedRecordPart;
myFixedID INT;

Function main()

dynamicPerson.ID = 123;
dynamicPerson.Job = "Student";

fixedPerson.ID = 456;
fixedPerson.Job = "Teacher";

relationship Dictionary
{
  dynamicRecord=dynamicPerson,
  staticRecord=fixedPerson
```

```

    };
  end
end
end

```

Se aplican las siguientes normas:

- Una referencia a un valor de diccionario es dinámica y cada referencia subordinada es dinámica. Considere el efecto en caso de que el código incluya las sentencias siguientes:

```

myDynamicID INT;
myDynamicID = relationship.dynamicRecord.ID;

```

La referencia a `dynamicRecord` sería dinámica y la referencia a `ID` sería dinámica, con el identificador *ID* visible en tiempo de ejecución.

- Una referencia que empieza por una estructura fija solo puede hacer referencia a la memoria interna de esa estructura. En el ejemplo actual, una referencia que empieza por `fixedPerson` puede acceder a los campos `ID` y `JOB` en el registro fijo, pero no puede acceder a otros campos.
- El código puede acceder dinámicamente a una estructura fija pero la misma sentencia de referencia no puede acceder a los campos de ese campo. En el ejemplo actual, la referencia siguiente no será válida porque el *ID* de identificador no está disponible en tiempo de ejecución:

```

myFixedID INT;

// NO válido
myFixedID = relationship.fixedRecord.ID;

```

Puede manejar el problema declarando otro registro fijo y asignándole valores del registro fijo que está en el diccionario:

```

myFixedID INT;
myOtherRecord myFixedRecordPart;
myOtherRecord = relationship.staticRecord;
myFixedID = myOtherRecord.ID;

```

El acceso dinámico es válido en las asignaciones (en los lados izquierdo o derecho); en expresiones lógicas y en las sentencias **set**, **for** y **openUI**.

Conceptos relacionados

“Sintaxis de corchete para el acceso dinámico” en la página 60
 “Diccionario” en la página 82
 “Componente de programa” en la página 139
 “Referencias a variables en EGL” en la página 58
 “Typedef” en la página 27

Tareas relacionadas

“Declarar variables y constantes en EGL” en la página 53

Consulta relacionada

“Asignaciones” en la página 374
 “Expresiones lógicas” en la página 497
 “Tipos primitivos” en la página 34
 “set” en la página 636

Reglas de ámbito y “this” en EGL

Si un componente EGL declara una variable o una constante, el identificador utilizado en la declaración está *en ámbito* (disponible) a lo largo de todo el componente:

- Si la declaración está en una función, el identificador está en el ámbito local de la función. Si la función `Function01` declara la variable `Var01`, por ejemplo, cualquier código de `Function01` puede hacer referencia a `Var01`. El identificador está disponible incluso en el código de función que precede a la declaración. La variable puede pasarse como un argumento a otra función, pero el identificador original no está disponible en esa función. El nombre del parámetro está disponible en la función de recepción porque el nombre del parámetro se declaró allí.
- Si la declaración está en un componente generable, como por ejemplo un programa, pero fuera de cualquier función, el identificador está en *ámbito global de programa*, lo que significa que cualquier función invocada por ese componente puede hacer referencia al identificador. Por ejemplo, si un programa declara `Var01` e invoca `Function01` que a su vez invoca `Function02`, `Var01` está disponible a lo largo de ambas funciones. Los identificadores de un texto o un formulario de impresión son globales para el componente generable que hace referencia al formulario. Esos identificadores están disponibles incluso en las funciones que preceden a la función que presenta el formulario.
- Si la declaración está en una biblioteca pero fuera de cualquier función, el identificador está en *ámbito de unidad de ejecución*, lo que significa global para todo el código de la unidad de ejecución.
- Los nombres de un `dataTable` y sus componentes pueden estar en ámbito global de programa, de unidad de ejecución o incluso aún mayor dependiendo del valor de las propiedades `dataTable` y del entorno en el que reside `dataTable`.

Los identificadores idénticos no pueden estar en el mismo ámbito. Sin embargo, la mayoría de identificadores hacen referencia a un área de memoria que está lógicamente dentro de un contenedor como por ejemplo un registro y, en esos casos, el código califica un identificador con el nombre del contenedor encerrador. Si la variable de función `myString` está en un registro llamado `myRecord01`, por ejemplo, el código hace referencia a la variable como un campo del registro:

```
myRecord01.myString
```

Si el mismo identificador está en dos ámbitos, cualquier referencia al identificador es una referencia al ámbito más local, pero puede utilizar calificadores para alterar temporalmente ese comportamiento:

- Considere el caso de un programa que declara la variable `Var01` e invoca una función que a su vez declara una variable del mismo nombre. Una referencia no calificada a `Var01` en la función implica el acceso a la variable declarada localmente.

Para acceder a un identificador que sea global de programa, incluso cuando un identificador local tenga precedencia, califique el identificador con la palabra clave *this*, como en el ejemplo siguiente:

```
this.Var01
```

En contadas ocasiones, la palabra clave *this* también se utiliza para alterar temporalmente un comportamiento de un bloque de valor establecido en una sentencia de asignación. Para conocer más detalles, consulte la sección *Establecer bloques de valor*.

- Considere el caso siguiente:
 - Un programa tiene una declaración de utilización para acceder a una biblioteca
 - El programa y la biblioteca declaran ambos una variable llamada `Var01`.

Si una función del programa incluye una referencia no calificada a Var01, la función accede a la variable del programa.

Para acceder a un identificador en ámbito de unidad de ejecución, incluso cuando otro identificador impida ese acceso, califique el identificador con el nombre de componente, como en el ejemplo siguiente (donde myLib es el nombre de una biblioteca):

```
myLib.Var01
```

Si la biblioteca o el dataTable están en un paquete distinto y no ha hecho referencia al componente en una sentencia de importación, el nombre del componente deberá ir precedido por el nombre del paquete, como en el ejemplo siguiente (donde myPkg es el nombre del paquete):

```
myPkg.myLib.Var01
```

El nombre del paquete siempre califica un nombre de componente y no puede ir inmediatamente precedido por una variable o un identificador de constante.

Finalmente, un identificador local puede ser igual que un nombre de biblioteca o dataTable si el identificador local está en un paquete distinto de aquel en el que reside el dataTable o la biblioteca. Para hacer referencia al nombre de biblioteca o dataTable, incluya el nombre del paquete.

Conceptos relacionados

“Componente de función” en la página 140

“Componente de biblioteca de tipo basicLibrary” en la página 142

“Componente de biblioteca de tipo basicLibrary” en la página 142

“PageHandler” en la página 194

“Componentes” en la página 17

“Componente de programa” en la página 139

“Referencias a componentes” en la página 21

“Referencias a variables en EGL”

“Visión general de las propiedades de EGL” en la página 64

“Estructura fija” en la página 26

“Typedef” en la página 27

Tareas relacionadas

“Declarar variables y constantes en EGL” en la página 53

Consulta relacionada

“Invocaciones de función” en la página 518

“Componente de función en formato fuente EGL” en la página 527

Referencias a variables en EGL

Para conocer detalles acerca de la distinción entre dos clases de acceso de memoria, consulte la sección *Acceso dinámico y estático*.

Independientemente de qué clase de acceso esté en vigor, la sintaxis con puntos de EGL normalmente es suficiente. Considere las definiciones de componente siguientes, por ejemplo:

```
Record myRecordPart01 type basicRecord
  myString    STRING;
  myRecordVar02 myRecordPart02;
end
```

```
Record myRecordPart02 type basicRecord
  myString02    STRING;
  myRecordVar03 myRecordPart03;
  myDictionary Dictionary
```

```

    {
      empnum=0005,
      lastName="Twain",
      firstName="Mark",
      birthday="021460"
    };
  end

  Record myRecordPart03 type basicRecord
    myInt INT;
    myDictionary Dictionary
    {
      customerNum=0005,
      lastName="Clemens"
    };
  end

```

Suponga que una función utiliza el componente de registro *myRecordPart01* como el tipo al declarar una variable llamada *myRecordVar01*.

Para consultar el campo *myInt*, liste los símbolos siguientes por orden:

- El nombre de la variable; en este caso, *myRecordVar01*
- Un punto (.)
- Una lista de los campos que llevan al campo de interés, con un punto para separar los identificadores entre sí, por ejemplo *myRecordVar02.myRecordVar03*
- El nombre del campo de interés, precedido de un punto, en este caso *.myInt*

La presencia de una matriz origina una ampliación directa de la misma sintaxis. Si *myRecordVar03* se declarará como una matriz de tres registros, por ejemplo, podría utilizar los símbolos siguientes para acceder al campo *myInt* en el tercer elemento de esa matriz:

```
myRecordVar01.myRecordVar02.myRecordVar03[3].myInt
```

La sintaxis con puntos también funciona cuando hace referencia a un campo de diccionario en este ejemplo. Para acceder al valor "Twain", especifique los caracteres siguientes en el lado derecho de una sentencia de asignación:

```
myRecordVar01.myRecordVar02.myDictionary.lastName
```

La presencia de un campo llamado *myDictionary* en dos componentes de registro distintos no supone un problema porque se hace referencia a cada campo del mismo nombre en relación a su propio registro encerrador.

También puede utilizar la sintaxis de puntos para hacer referencia a una constante (como por ejemplo *myConst*) en una biblioteca (como por ejemplo *myLib*):

```
myLib.myConstant
```

Hay otras dos sintaxis disponibles:

- Al utilizar el acceso dinámico, podrá especificar un nombre de campo como una serie entrecomillada o como un identificador de tipo *STRING*. Esta posibilidad se utiliza principalmente cuando está añadiendo o recuperando una entrada de diccionario (un par de clave y valor), en los casos siguientes:
 - La clave es una palabra reservada de *EGL* o incluye un carácter (como por ejemplo un punto o un espacio) que no es válido en un identificador
 - Desea utilizar una constante de serie para asignar o hacer referencia a la clave.

Para la sintaxis es necesario que coloque la variable, la constante o el literal dentro de un par de corchetes ([]). Los corchetes rellenos son equivalentes a un punto seguido de un identificador válido y puede mezclar las dos sintaxis. Sin embargo, el inicio de una referencia debe ser un identificador.

Por ejemplo, consulte la sección *Sintaxis de corchetes para el acceso dinámico*.

- Es posible que prefiera la comodidad de utilizar una sintaxis abreviada para hacer referencia a un campo en una estructura fija (dataTable, formulario de texto, formulario de impresión o registro fijo). Sin embargo, es recomendable que evite la utilización de esta sintaxis, en favor de la calificación completa descrita anteriormente.

Una sintaxis abreviada puede ser válida en relación con estructuras fijas solo si establece la propiedad **allowUnqualifiedItemReferences** en *yes*. Esa propiedad es característica de componentes de lógica generable como por ejemplo programas, bibliotecas y pageHandlers; el valor por omisión es *no*.

Para obtener más detalles, consulte *Sintaxis abreviadas para el acceso estático*.

Conceptos relacionados

"Sintaxis abreviada para referirse a estructuras fijas" en la página 62

"Sintaxis de corchete para el acceso dinámico"

"Acceso dinámico y estático" en la página 54

"Enumeraciones en EGL" en la página 484

"Componente de función" en la página 140

"Componentes" en la página 17

"Componente de programa" en la página 139

"Referencias a componentes" en la página 21

"Reglas de ámbito y "this" en EGL" en la página 56

"Estructura fija" en la página 26

"Typedef" en la página 27

Tareas relacionadas

"Declarar variables y constantes en EGL" en la página 53

Consulta relacionada

"Matrices" en la página 74

"Invocaciones de función" en la página 518

"Componente de función en formato fuente EGL" en la página 527

"Tipos primitivos" en la página 34

"Declaración use" en la página 954

Sintaxis de corchete para el acceso dinámico

Siempre que el acceso dinámico sea válido, puede hacer referencia a un campo utilizando una variable de serie, una constante o un literal entre corchetes. Cada par de corchetes relleno es equivalente a un punto seguido por un identificador válido.

Aunque algunas claves especificadas en una declaración de diccionario deben cumplir las reglas para los identificadores de EGL, puede especificar un rango más amplio de claves utilizando la sintaxis de corchetes en sentencias de asignación de EGL. La sintaxis de corchetes es necesaria en el ejemplo siguiente, donde se añaden dos entradas a un diccionario y se recupera el valor en cada una de esas entradas:

```
row Dictionary { lastname = "Smith" };
category, motto STRING;

row["Record"] ="Reserved word";
```

```

row["ibm.com"]="Think!";

category = row["Record"];
motto    = row["ibm.com"]

```

Si hace referencia a un valor utilizando un identificador mediante la sintaxis con puntos, puede hacer referencia al mismo valor utilizando una serie que sea equivalente al identificador. Las asignaciones siguientes tienen el mismo efecto:

```

row.age = 20;
row["age"] = 20;

```

Suponga que ha declarado un registro llamado `myRecordVar01` que incluye un campo llamado `myRecordVar02` y que `myRecordVar02` es en sí mismo un registro que incluye el diccionario anterior. Una referencia válida es la siguiente:

```
myRecordVar01.myRecordVar02.row.lastName
```

El acceso es estático para la mayoría de esa referencia. El acceso dinámico empieza cuando accede al campo en el diccionario. Suponga que estas constantes están en ámbito, sin embargo:

```

const SECOND STRING = "myRecordVar02";
const GROUP  STRING = "row";
const LAST   STRING = "lastName";

```

Puede codificar la referencia anterior de la forma siguiente:

```
myRecordVar01[SECOND][GROUP][LAST]
```

El primer símbolo de una referencia siempre debe ser un identificador válido, pero en este caso, el acceso dinámico entra en vigor después de ese identificador.

Puede mezclar ambas sintaxis: con puntos y con corchetes. Por ejemplo, la referencia siguiente es equivalente a la anterior:

```
myRecordVar01[SECOND].row[LAST]
```

Como un ejemplo final, considere una referencia con un índice de matriz:

```
myRecordVar01.myRecordVar02.myRecordVar03[3][2].myInt
```

Suponga que estas constantes están en ámbito:

```

const SECOND STRING = "myRecordVar02";
const THIRD  STRING = "myRecordVar03";
const CONTENT STRING = "myInt";

```

Puede codificar la referencia anterior de las siguientes maneras:

```
myRecordVar01[SECOND][THIRD][3][2][CONTENT]
```

```
myRecordVar01[SECOND][THIRD][3][2].myInt
```

```
myRecordVar01.myRecordVar02.THIRD[3][2][CONTENT]
```

Conceptos relacionados

“Sintaxis abreviada para referirse a estructuras fijas” en la página 62

“Acceso dinámico y estático” en la página 54

“Componente de función” en la página 140

“Componentes” en la página 17

“Componente de programa” en la página 139

“Referencias a componentes” en la página 21

“Referencias a variables en EGL” en la página 58

“Reglas de ámbito y “this” en EGL” en la página 56
“Estructura fija” en la página 26
“Typedef” en la página 27

Tareas relacionadas

“Declarar variables y constantes en EGL” en la página 53

Consulta relacionada

“Matrices” en la página 74
“Invocaciones de función” en la página 518
“Componente de función en formato fuente EGL” en la página 527
“Registros de opciones para registros MQ” en la página 665
“Tipos primitivos” en la página 34
“Declaración use” en la página 954

Sintaxis abreviada para referirse a estructuras fijas

Las reglas siguientes son válidas para hacer referencia a campos en un dataTable, un formulario de texto, un formulario de impresión o un registro fijo:

- Si hace referencia a un campo en un contenedor como un registro fijo, puede utilizar la sintaxis de puntos habitual para evitar la ambigüedad del área de memoria a la que se hace referencia. Por ejemplo, considere la siguiente declaración de componente:

```
Record myRecordPart type serialRecord
{
  fileName = "myFile"
}
10 myTop;
20 myNext;
30 myAlmost;
40 myChar CHAR(10);
40 myChar02 CHAR(10);
end
```

Suponga que una función utiliza el componente de registro *myRecordPart* como el tipo al declarar una variable llamada *myRecordVar*.

Una referencia válida a *myChar* en *myRecordVar* es la siguiente:

```
myRecordVar.myTop.myNext.myAlmost.myChar
```

Esta referencia se considera *totalmente calificada*.

- Si desea hacer referencia a un campo cuyo nombre es exclusivo dentro de una estructura, puede especificar el nombre de variable, seguido de un punto, seguido del nombre del campo. Las referencias válidas para el ejemplo anterior incluyen el siguiente símbolo:

```
myRecordVar.myChar
```

Esta referencia se considera *parcialmente calificada*.

No se puede calificar parcialmente un nombre de campo de ninguna otra forma. No se pueden incluir solamente algunos de los nombres de campo que están entre el nombre de variable y el nombre de campo de interés, por ejemplo, ni tampoco se puede eliminar el nombre de variable al mismo tiempo que se mantienen algunos de los nombres de campo de estructura que son superiores al campo de interés. Las siguientes referencias *no* son válidas para el ejemplo anterior:

```
// NO válido
myRecordVar.myNext.myChar
myRecordVar.myAlmost.myChar
myNext.myChar
myAlmost.myChar
```

- Puede hacer referencia a un campo sin preceder el nombre sin ningún calificador. Las referencias válidas para el ejemplo anterior incluyen los siguientes símbolos:

```
myChar
myChar02
```

Estas referencias se consideran *no calificadas*.

- Debe calificar cualquier referencia a un campo de estructura tanto como sea necesario para evitar cualquier ambigüedad.
- El nombre de un campo de estructura puede ser un asterisco (*) si el área de memoria relacionada es un *rellenador*, que es un área cuyo nombre no es importante. No se puede incluir un asterisco en una referencia. Considere el siguiente ejemplo:

```
record myRecordPart type serialRecord
{
  fileName = "myFile"
}
10 person;
20 *;
30 streetAddress1 CHAR(30);
30 streetAddress2 CHAR(30);
30 nation CHAR(20);
end
```

Si utiliza este componente como un tipo al declarar la variable *myRecordVar*, puede hacer referencia a *myRecordVar.nation* o *nation*, pero las siguientes referencias no son válidas:

```
// NO válido
myRecordVar.*.streetAddress1
myRecordVar.*.streetAddress2
myRecordVar.*.nation
```

- Cuando EGL intenta resolver una referencia, primero se busca en los nombres de variables locales, luego en los nombres de campos de estructura de los registros utilizados para la E/S de la misma función, después en los nombres de otros campos de estructura locales y finalmente en los nombres que son globales al programa.

Considere el caso en que una función declara una variable primitiva llamada *nation* y una variable que señala al siguiente registro básico:

```
record myRecordPart
10 myTop;
20 myNext;
30 nation CHAR(20);
end
```

Una referencia no calificada a *nation* hace referencia a la variable primitiva, no al campo de estructura.

- Una búsqueda en los nombres no muestra ninguna preferencia por las variables primitivas globales al programa frente a los campos de estructura globales al programa. Considere el caso en que un programa declara una variable primitiva llamada *nation* y una variable que señala al formato del siguiente registro básico:

```
record myRecordPart
10 myTop;
20 myNext;
30 nation CHAR(20);
end
```

Una referencia no calificada a *nation* no es satisfactoria ya que *nation* podría hacer referencia a la variable primitiva o al campo de estructura. Se puede hacer referencia al campo de estructura, pero sólo calificando la referencia.

Para obtener más información sobre normas, consulte las secciones *Matrices* y *Declaración de uso*.

Conceptos relacionados

- "Componente de función" en la página 140
- "Componentes" en la página 17
- "Componente de programa" en la página 139
- "Referencias a componentes" en la página 21
- "Referencias a variables en EGL" en la página 58
- "Reglas de ámbito y "this" en EGL" en la página 56
- "Estructura fija" en la página 26
- "Typedef" en la página 27

Tareas relacionadas

- "Declarar variables y constantes en EGL" en la página 53

Consulta relacionada

- "Matrices" en la página 74
- "Invocaciones de función" en la página 518
- "Componente de función en formato fuente EGL" en la página 527
- "Registros de opciones para registros MQ" en la página 665
- "Tipos primitivos" en la página 34
- "Declaración use" en la página 954

Visión general de las propiedades de EGL

La mayoría de componentes de EGL tienen un conjunto de propiedades que se utilizan para crear una salida adecuada durante la generación. El conjunto de propiedades válidas varía según el contexto:

- Cada tipo de componente define un conjunto de propiedades que son para el tipo de componente en su conjunto. Por ejemplo, cada componente de programa tiene una propiedad llamada **alias** que identifica el nombre de la unidad compilable.

Si un componente es un subtipo, hay más propiedades disponibles. Un programa del tipo **textUI** tiene una propiedad llamada **alias**, así como una propiedad llamada **inputForm**. La última identifica un formulario de texto que se presenta al usuario antes de ejecutar la lógica del programa.

- Muchos tipos de componentes también definen un conjunto de propiedades que se utilizarán en cualquiera de los campos primitivos que son componentes de dicho tipo de componente. Por ejemplo, un componente de registro de tipo **SQLRecord** incluye un conjunto de campos primitivos, cada uno de cuales tiene una propiedad **column** que identifica la columna de tabla SQL a la que accede el campo.

Las propiedades que están disponibles en un componente **DataItem** incluyen *todas* las propiedades de nivel de campo primitivo que son válidas en cualquier contexto. Considere, por ejemplo, un componente **DataItem** que representa un ID de nueve (y sólo nueve) dígitos, donde en algunos casos el ID se asocia con una columna de base de datos relacional llamada **SSN**:

```
DataItem IDPart CHAR(9)
{
    minInput = 9,        // requiere 9 caracteres de entrada
    isDigits = yes,     // requiere dígitos
    columnName = "SSN" // está relacionado con una columna
}
```

Puede declarar una variable de tipo **IDPart** de la manera siguiente:

```
myVariable IDPart;
```

Puede declarar esa variable en un componente compuesto como por ejemplo un componente de registro o directamente en un componente de lógica como por ejemplo un programa. En cada caso, el tipo de componente determina si se utiliza una determinada propiedad.

En el ejemplo actual, la propiedad **columnName** se utiliza solamente si la variable se declara en un registro de tipo SQLRecord. Las dos propiedades de validación solo se utilizan si la variable se declara en un componente de interfaz de usuario, como por ejemplo pageHandler.

- En algunas declaraciones de variable, puede alterar temporalmente una propiedad especificada en la definición de componente relacionada, pero solo si la propiedad resulta útil en el contexto en el que se declaró la variable:
 - La alteración temporal del contexto es posible cuando declara una variable basada en un componente DataItem. La sentencia siguiente declara un campo PageHandler del tipo SSN (tal como se ha definido anteriormente), pero para la sentencia no es necesario que el usuario teclee dígitos:

```
myVariable IDPart { isDigits = no };
```

En este ejemplo, la propiedad **minInput** no se ve afectada por la alteración temporal y se ignora la propiedad **columnName**.
 - En la mayoría de los casos, no es posible alterar temporalmente las propiedades de componentes compuestos, como por ejemplo componentes de registro.
- Cuando define una estructura fija, puede asignar propiedades a los campos de estructura elementales y puede alterar esas propiedades al declarar una variable relacionada. También puede asignar propiedades a un campo de estructura que tenga campos de estructura subordinados, pero en tales casos, las propiedades asignadas no se tienen en cuenta a menos que la documentación de la propiedad diga lo contrario.
- Cuando declara una variable de un tipo primitivo, puede establecer cualquiera de las propiedades de nivel de campo primitivo que sean útiles en el contexto de la declaración de variable.

No se puede acceder a una propiedad en tiempo de ejecución. Cuando crea variables basadas en un componente de registro SQL, por ejemplo, la lógica escrita no puede recuperar ni cambiar los nombres asignados a la propiedad **tableNames** que se identifica con las tablas SQL a las que se accede por registro. Incluso aunque altere temporalmente un valor de propiedad en una declaración de variable, la lógica no podrá cambiar el valor especificado durante el desarrollo.

La falta de acceso de tiempo de ejecución a un valor de propiedad significa que cuando asigna el contenido de una variable o utiliza la variable como un argumento, el valor de propiedad no se transfiere conjuntamente con el contenido. Si copia datos de un registro SQL a otro, por ejemplo, no se hacen cambios en la especificación de a qué tablas SQL accede el registro destino. Igualmente, cuando pasa un registro SQL a una función EGL, el parámetro recibe contenido de campo, pero retiene las especificaciones de tabla SQL que se asignaron durante el desarrollo.

Los componentes EGL predefinidos como por ejemplo ConsoleField pueden incluir propiedades y campos. Al contrario que las propiedades, los campos *están* disponibles en tiempo de ejecución. La lógica que escribe puede leer el valor del campo y en muchos casos, cambiarlo.

Un bloque de establecimiento de valor es un área de código en la que puede establecer los valores de propiedad y de campo. Para obtener detalles, consulte la sección *Bloque de establecimiento de valor*.

Conceptos relacionados

- “Referencias a variables en EGL” en la página 58
- “Bloques de establecimiento de valor” en la página 67

Consulta relacionada

- “Componente de formulario en formato fuente EGL” en la página 511
- “Propiedades de elementos SQL” en la página 67

Propiedades de presentación de campos

Las propiedades de presentación de campos de EGL especifican características que tienen relevancia cuando se visualiza un campo en una salida en pantalla, cuando el destino es una ventana de mandatos, pero no un navegador Web.

Las propiedades son las siguientes:

- “color” en la página 692
- “highlight” en la página 700
- “intensity” en la página 701
- “outline” en la página 709

Además, las siguientes propiedades tienen relevancia cuando el campo se visualiza en una salida imprimible, cuando el destino es una impresora o un archivo de impresión:

- La propiedad **highlight** (pero sólo para *underline* y *noHighlight*)
- La propiedad **outline**, que es adecuada solamente para los dispositivos que soportan los caracteres de doble byte

Las propiedades de presentación de campos no tienen efecto alguno sobre los datos que se devuelven al programa desde un formulario de texto; son únicamente para salida.

Propiedades de formato

Las propiedades de formato especifican características relevantes cuando se presentan datos en un formulario o un navegador Web:

- “align” en la página 690
- “currency” en la página 694
- “currencySymbol” en la página 694
- “dateFormat” en la página 695
- “fillCharacter” en la página 699
- “isBoolean” en la página 701
- “lineWrap” en la página 704
- “lowerCase” en la página 705
- “masked” en la página 705
- “numericSeparator” en la página 709
- “outline” en la página 709
- “sign” en la página 712
- “timeFormat” en la página 715

- “timeStampFormat” en la página 716
- “upperCase” en la página 717
- “zeroFormat” en la página 722

Conceptos relacionados

“Visión general de las propiedades de EGL” en la página 64

Propiedades de elementos SQL

Las propiedades de elementos SQL especifican las características que son relevantes al utilizar un elemento en un registro de tipo SQLRecord. Sin embargo, no es necesario especificar ninguna de las propiedades de elemento SQL, ya que se suministran valores por omisión.

Las propiedades son las siguientes:

- “column” en la página 693
- “isNullable” en la página 702
- “isReadOnly” en la página 703
- “maxLen” en la página 705
- “persistent” en la página 710
- “sqlDataCode” en la página 713
- “sqlVariableLen” en la página 714

Propiedades de validación

Las propiedades de validación limitan lo aceptable cuando el usuario especifica datos en un formulario de texto.

Las propiedades son las siguientes:

- “fill” en la página 699
- “inputRequired” en la página 700
- “inputRequiredMsgKey” en la página 700
- “isDecimalDigit” en la página 702
- “isHexDigit” en la página 702
- “minimumInput” en la página 706
- “minimumInputMsgKey” en la página 706
- “needsSOSI” en la página 707
- “typeChkMsgKey” en la página 716
- “validatorDataTable” en la página 718
- “validatorDataTableMsgKey” en la página 719
- “validatorFunction” en la página 719
- “validatorFunctionMsgKey” en la página 720
- “validValues” en la página 720
- “validValuesMsgKey” en la página 722

Bloques de establecimiento de valor

Un bloque de establecimiento de valor es un área de código en la que puede establecer los valores de propiedad y de campo. Para obtener más información, consulte la sección *Visión general de las propiedades de EGL*.

Un bloque de establecimiento de valor está disponible cuando se lleva a cabo cualquiera de las acciones siguientes:

- Definir un componente
- Declarar una variable
- Codificar un formulario especial de una sentencia de asignación
- Codificar una sentencia **openUI** tal como se describe en *openUI*

En los dos últimos casos, solo puede asignar valores a campos.

Nota: Se aplica una restricción a los campos de estructuras fijas. Puede utilizar bloques de establecimiento de valor para asignar los valores de las propiedades de nivel de campo primitivo, pero no para establecer los valores de los mismos campos.

Bloques de establecimiento de valor para situaciones elementales

Considere las reglas que se aplican en los casos más elementales:

- Cada bloque de establecimiento de valor empieza con una llave de apertura (**{**), incluye una lista de entradas separadas por comas o una sola entrada y finaliza con una llave de clausura (**}**)
- Las entradas están todas en dos formatos:
 - Cada entrada se compone de un par identificador y valor como por ejemplo **inputRequired = yes**
 - Cada entrada contiene valores que se asignan de forma posicional, cuando se asignan valores sucesivos a elementos sucesivos de una matriz.

En todos los casos, el bloque de establecimiento de valor está en el ámbito del componente, la variable o el campo que se modifica. Las variaciones de la sintaxis se ilustran mejor mediante un ejemplo.

El primer ejemplo muestra un componente `dataItem` que tiene dos propiedades (**inputRequired** y **align**) :

```
// el ámbito del bloque de establecimiento de valor es myPart
DataItem myPart INT
{
  inputRequired = yes,
  align = left
}
end
```

El ejemplo siguiente muestra una variable de tipo primitivo.

```
// el ámbito es myVariable
myVariable INT
{
  inputRequired = yes,
  align = left
};
```

El siguiente ejemplo muestra una declaración de componente de registro SQL, que incluye dos propiedades de registro (**tableNames** y **keyItems**):

```
// El ámbito es myRecordPart
Record myRecordPart type SQLRecord
{ tableNames = [{"myTable"}],
  keyItems = [{"myKey"} ]
myKey CHAR(10);
```

```

myOtherKey CHAR(10);
myContent01 CHAR(60);
myContent02 CHAR(60);
end

```

El ejemplo siguiente muestra una declaración de variable que utiliza el componente anterior como un tipo, altera temporalmente una de las dos propiedades de registro y establece dos campos en el registro:

```

// El ámbito es myRecord
myRecord myRecordPart
{
  keyItems = ["myOtherKey"],
  myContent01 = "abc",
  myContent02 = "xyz"
};

```

Los ejemplos adicionales incluyen declaraciones de variable y sentencias de asignación:

```

// el ejemplo muestra el único caso en el que
// puede alterarse temporalmente una propiedad de registro en una
// declaración de variable.
// el ámbito es myRecord
myRecord myRecordPart {keyItems = ["myOtherKey"]};

// el ámbito es myInteger, que es una matriz
myInteger INT[5] {1,2,3,4,5};

// estas sentencias de asignación
// no tiene bloques de establecimiento de valor
myRecord02.myContent01 = "abc";
myRecord02.myContent02 = "xyz";

// esta sentencia de asignación abreviada
// es equivalente a las dos anteriores y
// el ámbito es myRecord02
myRecord02
{
  myContent01="abc",
  myContent02="xyz"
};

// Esta sentencia de asignación abreviada
// restablece los primeros cuatro elementos de la matriz
// declarada anteriormente
myInteger{6,7,8,9};

```

La sentencia de asignación abreviada no está disponible para campos en una estructura fija.

Bloques de establecimiento de valor para un campo de un campo

Cuando está asignando valores para un campo de un campo, se utiliza una sintaxis en la que el bloque de establecimiento de valor está en un ámbito tal que las entradas solo están modificando el campo de interés.

Considere las definiciones de componente siguientes:

```

record myBasicRecPart03 type basicRecord
  myInt04 INT;
end

record myBasicRecPart02 type basicRecord

```

```

    myInt03 INT;
    myRec03 myBasicRecPart03;
end

record myBasicRecPart type basicRecord
    myInt01 INT;
    myInt02 INT;
    myRec02 myBasicRecPart02;
end

```

Puede asignar un valor de propiedad para cualquier campo de la manera siguiente:

- Cree un bloque de establecimiento de valor para el registro
- Incorpore una serie de nombres de campo para restringir el ámbito
- Cree el bloque de establecimiento de valor específico del campo

La sintaxis para asignar un valor de propiedad puede tener tres formas, tal como se muestra en los ejemplos siguientes, aplicables al campo myInt04:

```

// sintaxis con puntos, tal como se describe en
// Referencias a variables en EGL.
myRecB myBasicRecPart
{
    myRec02.myRec03.myInt04{ align = left }
};

// sintaxis con corchetes, tal como se describe en
// Sintaxis con corchetes para acceso dinámico.
// No puede utilizar esta sintaxis para
// campos en estructuras fijas.
myRecC myBasicRecPart
{
    myRec02["myRec03"]["myInt04"]{ align = left }
};

// sintaxis con llaves
myRecA myBasicRecPart
{
    myRec02 {myRec03 { myInt04 { align = left }}}
};

```

Incluso en casos complejos, se utiliza una coma para separar una entrada de la siguiente en un bloque de establecimiento de valor, pero tiene que tener en cuenta el nivel de anidación de un bloque dado:

```

// sintaxis con puntos
myRecB myBasicRecPart
{
    myInt01 = 4,
    myInt02 = 5,
    myRec02.myRec03.myInt04{ align = left },
    myRec02.myInt03 = 6
};

// sintaxis con corchetes
myRecC myBasicRecPart
{
    myInt01 = 4,
    myInt02 = 5,
    myRec02["myRec03"]["myInt04"]{ align = left },
    myRec02["myInt03"] = 6
};

// sintaxis con llaves;
// aunque esta utilización es mucho más difícil de mantener

```

```

myRecA myBasicRecPart
{
  myInt01 = 4,
  myInt02 = 5,
  myRec02
  {
    myRec03
    { myInt04
      { action = label5 }},
    myInt03 = 6
  }
};

```

Utilización de "this"

En una declaración de variable o una sentencia de asignación, puede tener un contenedor (como por ejemplo un registro SQL) que incluya un campo (como por ejemplo *keyItems*) cuyo nombre coincida con el de una propiedad de registro. Para referirse al campo en lugar de a la propiedad, utilice **this**, que establece el ámbito correcto para el bloque de establecimiento de valor o para una entrada en el bloque de establecimiento de valor.

Considere la declaración de registro siguiente:

```

Record myRecordPart type SQLRecord
{ tableNames = ["myTable"],
  keyItems = ["myKey"] }
myKey CHAR(10);
myOtherKey CHAR(10);
keyItems CHAR(60);
end

```

La declaración de registro siguiente establece primero un valor para la propiedad *keyItems*, y después establece un valor para el campo del mismo nombre:

```

myRecord myRecordPart
{
  keyItems = ["myOtherKey"],
  this.keyItems = "abc"
};

```

La sección siguiente proporciona un ejemplo adicional en una declaración de matriz.

Bloques de establecimiento de valor, matrices y elementos de matriz

Cuando declara una matriz dinámica, puede especificar el número inicial de elementos, como en este ejemplo:

```
col1 ConsoleField[5];
```

Las asignaciones de un bloque de establecimiento de valor hacen referencia a las propiedades y los campos predefinidos en cada uno de los elementos iniciales de tipo *ConsoleField*, aunque no a ningún elemento añadido posteriormente:

```

col1 ConsoleField[5]
{
  position = [1,1],
  color = red
};

```

Para asignar valores a un elemento determinado en una declaración de variable, cree un bloque de establecimiento de valor cuyo ámbito sea ese elemento. Tal como se muestra en el ejemplo siguiente, especifique el ámbito utilizando la palabra clave **this** con índice entre corchetes:

```
// asignar valores al segundo y cuarto elemento
coll ConsoleField[5]
{
  this[2] { color = blue },
  this[4] { color = blue }
};
```

Para conocer más detalles acerca de la utilización de la palabra clave **this**, consulte la sección *Ámbito de reglas y "this" en EGL*.

Puede utilizar entradas posicionales en un bloque de establecimiento de valor para asignar valores a elementos sucesivos de una matriz de cualquiera de los tipos siguientes (esto sólo es relevante al procesar informes o crear formularios de consola):

- ConsoleField
- Menu
- MenuItem
- Prompt
- Report
- ReportData

El ejemplo siguiente puede estar en una sentencia OpenUI. El ámbito de cada bloque de establecimiento de valor incorporado es un elemento de matriz específico:

```
new Menu
{
  labelText = "Universe",
  MenuItems =

  // valor de propiedad es una matriz dinámica
  [
    new MenuItem
    { name = "Expand",
      labelText = "Expand" },
    new MenuItem
    { name = "Collapse",
      labelText = "Collapse" }
  ]
}
```

Ejemplos adicionales

Observe los componentes siguientes:

```
Record Point
  x, y INT;
end

Record Rectangle
  topLeft, bottomRight Point;
end
```

El código siguiente es válido:

```
Function test()
  screen Rectangle
  {
```

```

    topLeft{x=1, y=1},
    bottomRight{x=80, y=24}
};

// cambie x, y en el código utilizando una sentencia
// equivalente al código siguiente:
// screen.topLeft.x = 1;
// screen.topLeft.y = 2;
screen.topLeft{x=1, y=2};
end

```

A continuación, inicialice una matriz de elementos dinámica de tipo Point en la misma función:

```

pts Point[2]
{
    this[1]{x=1, y=2},
    this[2]{x=2, y=3}
};

```

Establezca el valor de cada elemento que está ahora en la matriz y establezca el primer elemento en un valor distinto:

```

pts{ x=1, y=1 };
pts[1]{x=10, y=20};

```

En el ejemplo anterior, se utiliza pts[1] en lugar de this[1] porque el nombre de matriz no es ambiguo.

A continuación, observe otra matriz dinámica de tipo Point:

```

points Point[];

```

La sentencia de asignación siguiente no surte efecto porque no existen elementos:

```

points{x=1, y=1};

```

En contraste, la sentencia de asignación siguiente causa una excepción de fuera de límites porque se hace referencia a un elemento determinado que no existe:

```

points[1]{x=10, y=20};

```

Puede añadir elementos a la matriz y utilizar una sola sentencia para establecer valores en todos los elementos:

```

points.resize(2);
points{x=1, y=1};

```

Conceptos relacionados

“Sintaxis de corchete para el acceso dinámico” en la página 60

“Visión general de las propiedades de EGL” en la página 64

“Referencias a variables en EGL” en la página 58

“Reglas de ámbito y “this” en EGL” en la página 56

Consulta relacionada

“Matrices” en la página 74

“Inicialización de datos” en la página 471

“openUI” en la página 620

Matrices

EGL da soporte a los siguientes tipos de matrices:

- “Matrices dinámicas”
- “Matrices de campos de estructura” en la página 78

En cualquier caso, el número máximo de dimensiones soportadas es siete.

Matrices dinámicas

Cuando declara una matriz de registros, registros fijos o variables primitivas, la matriz tiene una identidad independiente de los elementos de la matriz:

- Hay un conjunto de funciones específicas de la matriz que permite aumentar o disminuir el número de elementos en tiempo de ejecución.
- La propiedad específica de la matriz **maxSize** indica cuántos elementos son válidos en la matriz. El valor por omisión no está enlazado, el número de elementos solo está limitado por los requisitos del entorno destino.

No es necesario especificar un número de elementos en la declaración pero, si lo hace, el número indica el número inicial de elementos. También puede especificar el número inicial de elementos listando una serie de constantes de matriz en la declaración, como puede hacerse solo con variables primitivas, no con registros.

La sintaxis para declarar una matriz dinámica se muestra en los ejemplos siguientes:

```
// Una matriz de 5 elementos o menos
myDataItem01 CHAR(30)[] { maxSize=5 };

// Una matriz de 6 elementos o menos,
// con 4 elementos inicialmente
myDataItem02 myDataItemPart[4] { maxSize=6 };

// Una matriz que no tiene elementos
// pero cuyo tamaño máximo es el mayor posible
myRecord myRecordPart[];

// Una matriz de 3 elementos a cuyos elementos
// se asignan los valores 1, 3 y 5
position int[] = [1,3,5];
```

Puede utilizar un entero literal para inicializar el número de elementos pero ni una variable ni una constante son válidas.

Cuando declara una matriz de matrices, un número inicial de elementos es válido en la dimensión especificada más a la izquierda y en cada dimensión subsiguiente hasta que a una dimensión le falte un número inicial. Las declaraciones siguientes son válidas:

```
// Válida, con maxsize indicando el máximo
// para la primera dimensión
myInt01 INT[3] [];
myInt02 INT[4][2] [] {maxsize = 12};
myInt03 INT[7][3][1];

// En el ejemplo siguiente, las constantes de matriz indican
// que la matriz externa tiene 3 elementos inicialmente.
// El primer elemento de la matriz externa es
// una matriz de dos elementos (con valores 1 y 2).
// El segundo elemento de la matriz externa es
// una matriz de tres elementos (con valores 3, 4, 5).
```

```
// El tercer elemento de la matriz externa es
// una matriz de dos elementos (con valores 6 y 7).
myInt04 INT[] [] = [[1,2],[3,4,5],[6,7]];
```

En el ejemplo siguiente, la sintaxis no es válida porque (por ejemplo) la matriz `myInt04` se declara como una matriz sin elementos pero a cada uno de esos elementos se le asignan 3 elementos:

```
// NO válido
myInt04 INT[] [3];
myInt05 INT[5] [] [2];
```

Una matriz especificada como un parámetro de función o programa no puede especificar el número de elementos.

Cuando el código hace referencia a una matriz o a un elemento de matriz, se aplican las siguientes normas:

- Un subíndice de elemento puede ser cualquier expresión numérica que dé como resultado un entero, pero la expresión no puede incluir una invocación de función.
- Si el código hace referencia a una matriz dinámica pero no especifica subíndices, la referencia se hace a la matriz en su totalidad.

Una situación de memoria insuficiente se trata como un error catastrófico y finaliza el programa.

Funciones de matriz dinámica

Para cada matriz dinámica existe un conjunto de funciones y variables de sólo lectura. En el ejemplo siguiente, la matriz se llama *series*:

```
series.resize(100);
```

El nombre de la matriz puede incluir un conjunto de corchetes, cada uno con un entero. A continuación se ofrece un ejemplo:

```
series INT[] [];
```

```
// redimensiona el segundo elemento
// de series, que es una matriz de matrices
series[2].resize(100);
```

En las secciones siguientes, sustituya el nombre de matriz por *nombreMatriz* y tenga en cuenta que el nombre puede calificarse mediante un nombre de paquete, un nombre de biblioteca o ambos.

appendAll():

```
nombreMatriz.appendAll(añadirMatriz Array in)
```

Esta función realiza lo siguiente:

- Se añade a la matriz a la que se hace referencia mediante *nombreMatriz*, añadiendo una copia de la matriz a la que se hace referencia mediante *añadirMatriz*
- Incrementa el tamaño de la matriz en el número de elementos añadidos
- Asigna un valor de índice apropiado a cada uno de los elementos añadidos

Los elementos de *añadirMatriz* deben ser del mismo tipo que los elementos de *nombreMatriz*.

appendElement()

`nombreMatriz.appendElement(contenido ArrayElement in)`

Esta función coloca un elemento al final de la matriz especificada e incrementa el tamaño en uno. Para *contenido*, puede sustituir una variable del tipo apropiado; como alternativa, puede especificar un literal que se asigna a un elemento creado durante la operación. El proceso copia datos; si asigna una variable, dicha variable todavía está disponible para fines de comparación u otros.

Las normas para asignar un literal se especifican en la sección *Asignaciones*.

getMaxSize()

`nombreMatriz.getMaxSize ()` returns (INT)

Esta función devuelve un entero que indica el número máximo de elementos permitidos en la matriz.

getSize()

`nombreMatriz.getSize ()` returns (INT)

Esta función devuelve un entero que indica el número de elementos dn la matriz. Es recomendable utilizar esta función en lugar de *SysLib.size* al trabajar con matrices dinámicas.

Otra función proporciona una funcionalidad equivalente a la de `nombreMatriz.getSize`:

`SysLib.size ()` returns (INT)

Sin embargo, es recomendable utilizar `nombreMatriz.getSize ()` al trabajar con matrices dinámicas.

insertElement()

`nombreMatriz.insertElement (contenido ArrayElement in, indice INT in)`

Esta función realiza lo siguiente:

- Coloca un elemento delante del elemento que ahora está en la ubicación especificada de la matriz
- Incrementa el tamaño de la matriz en uno
- Incrementa el índice de cada elemento que reside después del elemento insertado

contenido es el nuevo contenido (una constante o variable del tipo apropiado para la matriz) e *índice* es un literal entero o una variable numérica que indica la ubicación del nuevo elemento.

Si *índice* tiene un elemento más que el número de elementos de la matriz, la función crea un nuevo elemento al final de la matriz e incrementa el tamaño de la matriz en uno.

removeAll()

`nombreMatriz.removeAll ()`

Esta función elimina de la memoria los elementos de la matriz. La matriz puede utilizarse, pero su tamaño es cero.

removeElement()

`nombreMatriz.removeElement(índice INT in)`

Esta función elimina el elemento de la ubicación específica, disminuye el tamaño de la matriz en uno y disminuye el índice de cada elemento que reside después del elemento eliminado.

índice es un literal entero o una variable numérica que indica la ubicación del elemento que debe eliminarse.

resize()

`nombreMatriz.resize(tamaño INT in)`

Esta función aumenta o disminuye el tamaño actual de la matriz hasta el tamaño especificado en *tamaño* que es un literal entero, una constante o una variable. Si el valor de *tamaño* es mayor que el tamaño máximo permitido para la matriz, la unidad de ejecución termina.

reSizeAll()

`nombreMatriz.reSizeAll(tamaños INT[
] in)`

Esta función añade o disminuye cada dimensión de una matriz multidimensional. El parámetro *tamaños* es una matriz de enteros en la que cada elemento sucesivo especifica el tamaño de una dimensión sucesiva. Si el número de dimensiones redimensionadas es mayor que el número de dimensiones en *nombreMatriz* o si un valor de un elemento de *tamaños* es mayor que el tamaño máximo permitido en la dimensión equivalente de *nombreMatriz*, la unidad de ejecución termina.

setMaxSize()

`nombreMatriz.setMaxSize (tamaño INT in)`

La función establece el número máximo de elementos permitidos en la matriz. Si establece el tamaño máximo en un valor menor que el tamaño actual de la matriz, la unidad de ejecución termina.

setMaxSizes()

`nombreMatriz.setMaxSizes(tamaños INT[
] in)`

Esta función establece cada dimensión de una matriz multidimensional. El parámetro *tamaños* es una matriz de enteros en la que cada elemento sucesivo especifica el tamaño máximo de una dimensión sucesiva. Si el número de dimensiones especificado es mayor que el número de dimensiones de *nombreMatriz* o si un valor de un elemento de *tamaños* es menor que el número actual de elementos de la dimensión equivalente de *nombreMatriz*, la unidad de ejecución termina.

Utilización de matrices dinámicas como argumentos y parámetros

Una matriz dinámica puede pasarse como argumento a una función EGL. El parámetro relacionado debe definirse como una matriz dinámica del mismo tipo que el argumento; y para un elemento de datos, el tipo debe incluir la misma longitud y posiciones decimales, si existen.

Una matriz dinámica no puede pasarse como argumento a otro programa.

A continuación se ofrece un ejemplo de una función que utiliza una matriz dinámica como parámetro:

```
Function getAll (employees Employee[])  
;  
end
```

Durante la ejecución, el tamaño máximo de un parámetro es el tamaño máximo declarado para el argumento correspondiente. La función o programa llamado puede cambiar el tamaño de la matriz y el cambio se aplica en el código invocante.

Proceso SQL y matrices dinámicas

EGL permite utilizar una matriz dinámica para acceder a las filas de una base de datos relacional. Para obtener información detallada sobre cómo leer varias filas, consulte la sección *get*. Para obtener información detallada sobre cómo añadir varias filas, consulte la sección *add*.

Matrices de campos de estructura

Una matriz de campos de estructura se declara cuando se especifica que un campo de una estructura fija tiene un valor de aparición mayor que uno, como en el ejemplo siguiente:

```
Record myFixedRecordPart  
  10 mySi CHAR(1)[3];  
end
```

Si un registro fijo llamado *myRecord* se basa en dicho componente, el símbolo *myRecord.mySi* hace referencia a una matriz unidimensional de tres elementos, cada uno de los cuales es un carácter.

Utilización de una matriz de campos de estructura

Puede hacer referencia a toda una matriz de campos de estructura (por ejemplo, *myRecord.mySi*) en los siguientes contextos:

- Como segundo operando que utiliza un operador *in*. El operador prueba si un determinado valor está contenido en la matriz.
- Como parámetro de la función **sysLib.size**. Esta función devuelve el valor de apariciones del campo de estructura.

Un elemento de matriz que no es en sí mismo una matriz es un campo como cualquier otro, y se puede hacer referencia a dicho campo de varias maneras; por ejemplo, en una sentencia assignment o como argumento en una invocación de función.

Un subíndice de elemento puede ser cualquier expresión numérica que dé como resultado un entero, pero la expresión no puede incluir una invocación de función.

Matriz de campo de estructura unidimensional

Puede hacer referencia a un elemento de una matriz unidimensional, como por ejemplo *myRecord.mySi*, utilizando el nombre de la matriz seguido de un subíndice entre corchetes. El subíndice es un entero o un campo que se resuelve en un entero; por ejemplo, puede hacer referencia al segundo elemento de la matriz de ejemplo como *myStruct.mySi[2]*. El subíndice puede variar de 1 al valor de apariciones del campo de estructura, y se produce un error de entorno de ejecución si el subíndice está fuera de este rango.

Si utiliza el nombre de una matriz de campos de estructura en un contexto que requiere un campo, pero no especifica un subíndice entre corchetes, EGL presupone que hace referencia al primer elemento de la matriz, pero sólo si está en

modalidad de compatibilidad de VisualAge Generator. Es aconsejable identificar cada elemento explícitamente. Si no está en modalidad de compatibilidad de VisualAge Generator, será necesario que identifique cada elemento explícitamente.

Los ejemplos siguientes muestran cómo hacer referencia a elementos de una matriz unidimensional. En estos ejemplos, `valueOne` se resuelve en 1 y `valueTwo` se resuelve en 2:

```
// estos ejemplos hacen referencia al primero de tres elementos:
myRecord.mySi[valueOne]

// no recomendado y válido
// sólo si la compatibilidad de
// VisualAge Generator está en vigor:
myRecord.mySi

// este ejemplo hace referencia al segundo elemento:
myRecord.mySi[valueTwo]
```

Una matriz unidimensional puede tener subestructuras, como en el siguiente ejemplo:

```
record myRecord01Part
  10 name[3];
    20 firstOne CHAR(20);
    20 midOne CHAR(20);
    20 lastOne CHAR(20);
end
```

Si un registro llamado `myRecord01` se basa en el componente anterior, el símbolo `myRecord01.name` hace referencia a una matriz unidimensional de tres elementos, cada uno de los cuales tiene 60 caracteres y la longitud de `myRecord01` es 180.

Puede hacer referencia a cada elemento de `myRecord01.name` sin hacer referencia a la subestructura; por ejemplo, `myRecord01.name[2]` hace referencia al segundo elemento. También puede hacer referencia a una subestructura dentro de un elemento. Si, por ejemplo, se cumplen las normas de exclusividad, puede hacer referencia a los 20 últimos caracteres del segundo elemento de las siguientes maneras:

```
myRecord01.name.lastOne[2]
myRecord01.lastOne[2]
lastOne[2]
```

Los dos últimos son válidos solo si la propiedad de componente generable **allowUnqualifiedItemReferences** se establece en *yes*.

Para obtener detalles sobre los diferentes tipos de referencias, consulte la sección *Referencias a variables y constantes*.

Matriz de campos de estructura multidimensional

Si un elemento de estructura con un valor de apariciones mayor que uno tiene subestructuras y si una estructura subordinada también tiene un valor de apariciones mayor que uno, el elemento de estructura subordinada declara una matriz con una dimensión adicional.

Consideremos otro componente de registro:

```
record myRecord02Part
  10 siTop[3];
    20 siNext CHAR(20)[2];
end
```

Si un registro llamado *myRecord02* se basa en dicho componente, cada elemento de la matriz unidimensional *myRecord02.siTop* es en sí mismo una matriz unidimensional. Por ejemplo, puede hacer referencia a la segunda de las tres matrices unidimensionales subordinadas como *myRecord02.siTop[2]*. El elemento de estructura *siNext* declara una matriz bidimensional, y se puede hacer referencia un elemento de dicha matriz mediante cualquiera de estas sintaxis:

```
// fila 1, columna 2.
// la sintaxis siguiente se recomienda encarecidamente
// porque también funciona con matrices dinámicas
myRecord02.siTop[1].siNext[2]

// la sintaxis siguiente está soportada a efectos
// de compatibilidad con VisualAge Generator
myRecord02.siTop.siNext[1,2]
```

Para aclarar a qué área de memoria se hace referencia, consideremos cómo se almacenan los datos en una matriz multidimensional. En el ejemplo actual, *myRecord02* constituye 120 bytes. El área a la que se hace referencia se divide en una matriz unidimensional de tres elementos, cada uno de los cuales tiene 40 bytes:

```
siTop[1]      siTop[2]      siTop[3]
```

Cada elemento de la matriz unidimensional se vuelve a subdividir en una matriz de dos elementos, de 20 bytes cada uno, en la misma área de memoria:

```
siNext[1,1] siNext[1,2] siNext[2,1] siNext[2,2] siNext[3,1] siNext[3,2]
```

Una matriz bidimensional se almacena en orden de fila principal. Una implicación es que si se inicializa una matriz en un bucle *while* doble, se obtiene un mayor rendimiento procesando las columnas de una fila antes de procesar las columnas de una segunda fila:

```
// i, j, myTop0ccurs y myNext0ccurs son elementos de datos;
// myRecord02 es un registro; y
// sysLib.size() devuelve el valor de apariciones de un elemento de estructura.
i = 1;
j = 1;
myTop0ccurs = sysLib.size(myRecord02.siTop);
myNext0ccurs = sysLib.size(myRecord02.siTop.siNext);
while (i <= myTop0ccurs)
  while (j <= myNext0ccurs)
    myRecord02.siTop.siNext[i,j] = "abc";
    j = j + 1;
  end
  i = i + 1;
end
```

Debe especificar un valor para cada dimensión de una matriz multidimensional. La referencia *myRecord02.siTop.siNext[1]*, por ejemplo, no es válida para una matriz bidimensional.

A continuación se ofrece una declaración de ejemplo de una matriz tridimensional:

```
record myRecord03Part
  10 siTop[3];
  20 siNext[2];
  30 siLast CHAR(20) [5];
end
```

Si un registro llamado *myRecord03* se basa en dicho componente y se cumplen las normas de exclusividad, puede hacer referencia al último elemento de la matriz de las siguientes maneras:

```

// se muestra cada nivel y hay un
// subíndice en cada nivel, tal como se recomienda.
myRecord03.siTop[3].siNext[2].siLast[5]

// se muestra cada nivel y los subíndices están en niveles inferiores
myRecord03.siTop.siNext[3,2].siLast[5]
myRecord03.siTop.siNext[3][2].siLast[5]

// se muestra cada nivel y los subíndices están en el nivel más bajo
myRecord03.siTop.siNext.siLast[3,2,5]
myRecord03.siTop.siNext.siLast[3,2][5]
myRecord03.siTop.siNext.siLast[3][2,5]
myRecord03.siTop.siNext.siLast[3][2][5]

// se muestran el contenedor y el último nivel, con subíndices
myRecord03.siLast[3,2,5]
myRecord03.siLast[3,2][5]
myRecord03.siLast[3][2,5]
myRecord03.siLast[3][2][5]

// solo se muestra el último nivel, con subíndices
siLast[3,2,5]
siLast[3,2][5]
siLast[3][2,5]
siLast[3][2][5]

```

Tal como indica el ejemplo anterior, puede hacer referencia a un elemento de una matriz multidimensional añadiendo un conjunto de subíndices entre corchetes de cualquiera de las maneras posibles. En todos los casos, el primer subíndice hace referencia a la primera dimensión, el segundo subíndice hace referencia a la segunda dimensión y así sucesivamente. Cada subíndice puede variar de 1 al valor de apariciones del elemento de estructura relacionado, y se produce un error de entorno de ejecución si un subíndice se resuelve en un número que está fuera de dicho rango.

Primero, considere la situación en la que no hay subíndices implicados:

- Puede especificar una lista que empieza por el nombre de la variable y continúa con los nombres de elementos de estructura subordinada, con cada nombre separado del siguiente por un punto, como en este ejemplo:
`myRecord03.siTop.siNext.siLast`
- Puede especificar el nombre de la variable, seguido por un punto, seguido por el nombre del elemento de interés de nivel más bajo, como en este ejemplo:
`myRecord03.siLast`
- Si el elemento de interés de nivel más bajo es exclusivo en un espacio de nombres dado, puede especificar solamente ese elemento, como en este ejemplo:
`siLast`

A continuación, considere las reglas para colocar subíndices de matriz:

- Puede especificar un subíndice en cada nivel en el que uno de varios elementos sea válido, como en este ejemplo:
`myRecord03.siTop[3].siNext[2].siLast[5]`
- Puede especificar una serie de subíndices en cualquier nivel en el que uno de varios elementos sea válido, como en este ejemplo:
`myRecord03.siTop.siNext[3,2].siLast[5]`
- Puede especificar una serie de subíndices en cualquier nivel en el que uno de varios elementos sea válido, como en este ejemplo:
`myRecord03.siTop.siNext.siLast[3,2,5]`

- Si asigna más subíndices de los que corresponde en un nivel dado, se produce un error, como en el siguiente ejemplo:

```
// NO válido
myRecord03.siTop[3,2,5].siNext.siLast
```

- Puede aislar un subíndice en un corchete o puede visualizar una serie de subíndices, cada uno separado del siguiente por una coma o puede combinar los dos usos. Los ejemplos siguientes son válidos:

```
myRecord03.siTop.siNext.siLast[3,2,5]
myRecord03.siTop.siNext.siLast[3,2][5]
myRecord03.siTop.siNext.siLast[3][2,5]
myRecord03.siTop.siNext.siLast[3][2][5]
```

Conceptos relacionados

“Compatibilidad con VisualAge Generator” en la página 439

“Referencias a variables en EGL” en la página 58

Consulta relacionada

“add” en la página 561

“Asignaciones” en la página 374

“Límites de sistema EGL” en la página 494

“get” en la página 585

“operador in” en la página 532

“size()” en la página 909

Diccionario

Un *componente de diccionario* es un componente que está siempre disponible; no se define. Una variable basada en un componente de diccionario puede incluir un conjunto de claves y los valores relacionados y puede añadir y eliminar entradas de clave y valor en tiempo de ejecución. Las entradas se tratan como campos en un registro.

A continuación se proporciona un ejemplo de declaración de diccionario:

```
row Dictionary
{
  ID          = 5,
  lastName   = "Twain",
  firstName  = "Mark",
};
```

Cuando incluye entradas en la declaración, cada nombre de clave es un identificador EGL que debe ajustarse al convenio de denominación de EGL. Cuando añade entradas en tiempo de ejecución, tiene más flexibilidad; puede especificar un literal de serie, una constante o una variable y en ese caso, el contenido puede ser una palabra reservada de EGL o puede incluir caracteres que no serían válidos en un identificador. Para conocer más detalles, consulte la sección *Sintaxis de corchetes para el acceso dinámico*.

A continuación se ofrecen ejemplos de asignaciones:

```
row.age = 30;
row["Credit"] = 700;
row["Initial rating"] = 500
```

Si intenta asignar una clave que ya existe, se altera temporalmente la entrada de clave y valor existente. La asignación siguiente es válida y sustituye “Twain” por “Clemens”:

```
row.lastname = "Clemens";
```

Las asignaciones también pueden utilizarse para la recuperación de datos:

```
lastname String
age, credit, firstCredit int;

lastname = row.lastname;
age = row.age;
credit = row.credit;
credit = row["Credit"];
firstCredit = row["Initial rating"];
```

El valor en una entrada de clave y valor es de tipo ANY, lo que significa que puede poner distintos tipos de información en un solo diccionario. Cada valor puede ser uno de estos:

- Un registro declarado previamente u otra variable
- Una constante o un literal

Al poner una variable en un diccionario se asigna una copia de la variable. Considere el componente de registro siguiente:

```
Record myRecordPart
  x int;
end
```

El código siguiente coloca una variable de tipo myRecordPart en el diccionario y después cambia un valor en la variable original:

```
testValue int;

myRecord myRecordPart;

// establece un valor de variable y coloca
// una copia de la variable en el diccionario.
myRecord.x = 4;
row Dictionary
{
  theRecord myRecord;
}

// Coloca un valor nuevo en el registro original.
myRecord.x = 700;

// Accede a la copia del diccionario del registro,
// asignando 4 a testValue.
testValue = row.theRecord.x;
```

Al asignar un diccionario a otro se sustituye el contenido del destino por el contenido origen y se alteran temporalmente los valores de las propiedades del diccionario destino que se describen posteriormente. La sentencia condicional en el código siguiente es verdadera, por ejemplo:

```
row Dictionary { age = 30 };

newRow Dictionary { };
newRow = row

// se resuelve en true
if (newRow.age == 30)
;
end
```

Un conjunto de propiedades en la declaración afectan al proceso del diccionario. Un conjunto de funciones específicas de diccionario proporcionan datos y servicios al código.

Propiedades de diccionario

Cada entrada de propiedad y valor es sintácticamente equivalente a una entrada de clave y valor, tal como se muestra en este ejemplo y las entradas pueden estar en cualquier orden:

```
row Dictionary
{
  // propiedades
  caseSensitive = no,
  ordering = none,

  // campos
  ID          = 5,
  lastName   = "Twain",
  firstName  = "Mark"
  age = 30;
};
```

El código no puede añadir ni recuperar una propiedad ni el valor correspondiente. En el caso improbable de que desee utilizar un nombre de propiedad como una clave, utilice el nombre de variable como calificador cuando especifique la clave o haga referencia a ella, como en este ejemplo:

```
row Dictionary
{
  // propiedades
  caseSensitive = no,
  ordering = none,

  // campos
  row.caseSensitive = "yes"
  row.ordering = 50,
  age = 30
};
```

Las propiedades son las siguientes:

caseSensitive

Indica si la recuperación de una clave o del valor relacionado se ve afectada por el hecho de que la clave con la que se almacenó ese valor esté en mayúsculas o minúsculas. Las opciones son las siguientes:

No (el valor por omisión)

El acceso de clave no se ve afectado por las mayúsculas/minúsculas de la clave y las sentencias siguientes son equivalentes:

```
age = row.age;
age = row.AGE;
age = row["aGe"];
```

Yes

Las sentencias siguientes pueden tener distintos resultados, incluso aunque EGL sea principalmente un lenguaje no sensible a las mayúsculas/minúsculas:

```
age = row.age;
age = row.AGE;
age = row["aGe"];
```

El valor de la propiedad **caseSensitive** afecta al comportamiento de varias de las funciones descrita en una sección posterior.

ordering

Indica cómo deben ordenarse las entradas de clave y valor a efectos de

recuperación. El valor de esta propiedad afecta al comportamiento de las funciones **getKeys** y **getValues**, tal como se describe en una sección posterior.

Las opciones son las siguientes:

None (valor por omisión)

El código no puede fiarse del orden de las entradas de clave y valor.

Cuando el valor de la propiedad **ordering** es **None**, el orden de las claves (cuando se invoca la función **getKeys**) puede no ser igual que el orden de los valores (cuando se invoca la función **getValues**).

ByInsertion

Los pares de clave y valor están disponibles por el orden de inserción. Se considera que las entradas de la declaración se insertan primero, por orden de izquierda a derecha.

ByKey

Los pares de clave y valor están disponibles por orden de clave.

Funciones de diccionario

Para invocar las funciones siguientes, califique el nombre de función con el nombre del diccionario, como en este ejemplo en el que el diccionario se llama *row*:

```
if (row.containsKey(age))  
;  
end
```

containsKey()

dictionaryName.containsKey(clave **String** in) returns (**Boolean**)

Esta función devuelve true o false, dependiendo de si la serie de entrada (*clave*) es una clave en el diccionario. Si la propiedad del diccionario **caseSensitive** se establece en no, no se tienen en cuenta las mayúsculas/minúsculas, de lo contrario la función busca una coincidencia exacta, incluyendo las mayúsculas/minúsculas.

containsKey solo se utiliza en una expresión lógica.

getKeys()

dictionaryName.getKeys () returns (**String[]**)

Esta función devuelve una matriz de series, cada una de las cuales es una clave en el diccionario.

Si la propiedad del diccionario **caseSensitive** se establece en no, cada clave devuelta está en minúsculas, de lo contrario cada clave devuelta está en mayúsculas o minúsculas según lo estuviera la clave en la que estaba almacenada.

Si la propiedad de diccionario **ordering** se establece en no, no puede fiarse del orden de las claves devueltas, de lo contrario el orden es el especificado en la descripción de esa propiedad.

getValues()

dictionaryName.getValues () returns (**ANY[]**)

Esta función devuelve una matriz de valores de cualquier tipo. Cada valor está asociado con una clave del diccionario.

insertAll()

nombreDiccionario.insertAll(diccionarioOrigen Dictionary in)

Esta función actúa como si una serie de sentencias de asignación copiara las entradas de clave y valor del diccionario de origen (*diccionarioOrigen*) al *destino*, que es el diccionario cuyo nombre califica el nombre de función.

Si una clave está en el origen y no en el destino, se copia la entrada de clave y valor en el destino. Si una clave está tanto en el origen como en el destino, el valor de la entrada origen altera temporalmente la entrada en el destino. La determinación de si una clave del destino coincide con una del origen se ve afectada por el valor de la propiedad **caseSensitive** en cada diccionario.

Esta función es distinta de la asignación de un diccionario a otro porque la función **insertAll** retiene estas entradas:

- Las entradas de propiedad y valor en el destino
- Las entradas de clave y valor que están en el destino pero no en el origen.

removeElement()

nombreDiccionario.removeElement(clave String in)

Esta función elimina la entrada cuya serie de entrada (*clave*) es una clave del diccionario. Si la propiedad del diccionario **caseSensitive** se establece en no, no se tienen en cuenta las mayúsculas/minúsculas, de lo contrario la función busca una coincidencia exacta, incluyendo las mayúsculas/minúsculas.

removeAll()

nombreDiccionario.removeAll()

Esta función elimina todas las entradas de clave y valor del diccionario pero no tiene efecto sobre las propiedades del diccionario.

size()

nombreDiccionario.size() returns (INT)

Devuelve un entero que indica el número de entradas de clave y valor en el diccionario.

Conceptos relacionados

“Componentes” en la página 17

“Referencias a variables en EGL” en la página 58

Consulta relacionada

“Expresiones lógicas” en la página 497

ArrayDictionary

Un *componente arrayDictionary* es un componente que siempre está disponible, no se define. Una variable basada en un componente *arrayDictionary* permite acceder a una serie de matrices recuperando el mismo elemento numerado de cada matriz. Un conjunto de elementos recuperado de esta forma es en sí mismo un diccionario, con cada uno de los nombres de matriz originales tratado como una clave emparejada con el valor contenido en el elemento de matriz.

Un `arrayDictionary` resulta especialmente útil en relación con la tecnología de visualización descrita en la sección *Interfaz de usuario de consola*.

El gráfico siguiente muestra un `arrayDictionary` cuya declaración incluye matrices denominadas *ID*, *lastname*, *firstname* y *age*. La elipse encierra un diccionario que incluye las entradas de clave y valor siguientes:

```
ID = 5,  
lastName = "Twain",  
firstName = "Mark",  
age = 30
```

ID	Apellido	Nombre	Edad
5	Twain	Mark	30

La matriz de interés es la matriz de diccionarios, que muestra los diccionarios uno encima de otro en lugar de uno junto a otro. Sin embargo, la declaración de `arrayDictionary` necesita una lista inicial de matrices, las cuales se muestran una junto a otra.

El código siguiente muestra la declaración de una lista de matrices, seguida por la declaración de un `arrayDictionary` que utiliza esas matrices:

```
ID      INT[4];  
lastName STRING[4];  
firstName STRING[4];  
age     INT[4];  
  
myRows ArrayDictionary  
{  
  col1 = ID,  
  col2 = lastName,  
  col3 = firstName,  
  col4 = age  
};
```

Para recuperar valores, el código utiliza una sintaxis que aísla un diccionario determinado y después un campo determinado (una entrada de clave y valor) de ese diccionario. No puede utilizar la sintaxis `arrayDictionary` para actualizar un valor ni para cambiar ninguna característica del `arrayDictionary`.

En primer lugar, declare un diccionario y asigne una fila `arrayDictionary` a ese diccionario, como en este ejemplo:

```
row Dictionary = myRows[2];
```

A continuación, declare una variable del tipo adecuado y asigne un elemento a esa variable, como en cualquiera de estos ejemplos:

```
cell INT = row["ID"];
```

```
cell INT = row.ID;
```

Una sintaxis alternativa recupera el valor en un paso, como en cualquiera de estos ejemplos:

```
cell int = myRows[2]["ID"];
```

```
cell int = myRows[2].ID;
```

Conceptos relacionados

“Interfaz de usuario de consola” en la página 177

“Diccionario” en la página 82

“Referencias a variables en EGL” en la página 58

Sentencias EGL

Cada función EGL se compone de entre cero y muchas sentencias EGL de los siguientes tipos:

- Una *declaración de variable* o *declaración de constante* proporciona acceso a un área de memoria determinada. El valor de una variable puede cambiarse durante la ejecución; no así el valor de una constante. Cada tipo de declaración puede encontrarse en cualquier lugar de una función, excepto en un *bloque*, como se describe más adelante.
- Una *invocación de función* dirige el proceso a una función, como en este ejemplo:

```
myFunction(myInput);
```

Las llamadas recursivas sólo son válidas.

- Una *sentencia de asignación* puede copiar cualquiera de los siguientes valores en una variable:
 - Datos de una constante o variable
 - Un literal
 - Un valor devuelto desde una invocación de función
 - El resultado de un cálculo aritmético
 - El resultado de una concatenación de series

Ejemplos de sentencias de asignación:

```
myItem = 15;  
myItem = readFile(myKeyValue);  
myItem = bigValue - 32;  
record1.message = "Operación " + "satisfactoria";
```

- Una *sentencia de palabra clave* proporciona funciones adicionales, como por ejemplo acceso a archivos. Cada una de estas sentencias se nombra para la palabra clave que inicia la sentencia; por ejemplo:

```
add record1; // una sentencia add  
return (0); // una sentencia return
```

- Una *sentencia nula* es un punto y coma que no tiene ningún efecto, pero que puede ser de utilidad como espacio reservado, como en este ejemplo:

```
if (myItem == 5)  
    ; // sentencia nula  
else  
    myFunction(myItem);  
end
```

Las sentencias EGL no nulas tienen las siguientes características:

- Una sentencia puede hacer referencia a áreas de memoria determinadas, que son de los siguientes tipos:
 - Form
 - PageHandler
 - Record
 - DataTable
 - Item (una categoría que incluye elementos de datos, así como elementos de estructura de registros, formularios y tablas)
 - Array (un área de memoria basada en un elemento de estructura que tiene un valor de apariciones mayor que 1)
- Una sentencia puede incluir estos tipos de expresiones:
 - Una *expresión de fecha y hora* se resuelve en una fecha, un entero, un intervalo o una indicación de la hora
 - Una *expresión lógica* se resuelve en true o false
 - Una *expresión numérica* se resuelve en un número, que puede tener signo e incluir una coma decimal
 - Una *expresión de serie* se resuelve en una serie de caracteres, que puede incluir caracteres de un solo byte, caracteres de doble byte o una combinación de ambos
- Una sentencia finaliza con un punto y coma o con un *bloque*, que es una serie de cero o más sentencias subordinadas que actúan como una unidad. Las sentencias que contienen bloques finalizan con un delimitador de finalización, como en este ejemplo:

```

if (record2.status= "Y")
  record1.total = record1.total + 1;
  record1.message = "Operación satisfactoria";
else
  record1.message = "Operación anómala";
end

```

Un punto y coma después de un delimitador de finalización no es un error, pero se trata como sentencia nula.

Los nombres de sentencias y en todo EGL *no son sensibles* a mayúsculas y minúsculas; por ejemplo, *record1* es idéntico a *RECORD1* y tanto *add* como *ADD* hacen referencia a la misma palabra clave.

Nota: Al utilizar la pestaña de código fuente en Page Designer, puede enlazar manualmente componentes de un archivo JSP (específicamente de un archivo JavaServer Faces) con áreas de datos de un PageHandler. Aunque EGL no es sensible a las mayúsculas y minúsculas, los nombres de variable EGL a los que se hace referencia en el archivo JSP deben coincidir en cuanto a mayúsculas y minúsculas con la declaración de variable EGL; si la coincidencia no es total, se produce un error de JavaServer Faces. Es recomendable no cambiar las mayúsculas y minúsculas de una variable EGL después de enlazar esa variable con un campo JSP.

Las *palabras del sistema* son un conjunto de palabras que proporcionan funciones especiales:

- Una *función de sistema* ejecuta código y puede devolver un valor; por ejemplo:
 - **sysLib.minimum(arg1, arg2)** devuelve el mínimo de dos números
 - **strLib.strLen(arg1)** devuelve la longitud de una serie de caracteres

El calificador (**mathLibstrLib** o **sysLib**) sólo es necesario si el programa tiene una función con el mismo nombre.

- Una *variable de sistema* proporciona un valor sin invocar a una función; por ejemplo:
 - **sysVar.errorCode** contiene un código de estado después de que el programa acceda a un archivo y en otras situaciones
 - **sysVar.sqlcode** contiene un código de estado después de que el programa acceda a una base de datos relacional

El calificador **sysVar** sólo es necesario si el programa tiene una variable con el mismo nombre.

Una línea de una función puede contener más de una sentencia. Sin embargo, no es aconsejable incluir más de una sentencia por línea, ya que puede utilizar el depurador EGL para establecer un punto de interrupción sólo en la primera sentencia de una línea.

Consulte también el apartado *Comentarios*.

Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13

“Componente de función” en la página 140

“Componentes” en la página 17

Consulta relacionada

“add” en la página 561

“Asignaciones” en la página 374

“call” en la página 563

“case” en la página 566

“close” en la página 568

“Comentarios” en la página 438

“Inicialización de datos” en la página 471

“delete” en la página 571

“Palabras reservadas EGL” en la página 486

“execute” en la página 574

“Invocaciones de función” en la página 518

“get” en la página 585

“get next” en la página 597

“get previous” en la página 602

“if, else” en la página 608

“Palabras clave por orden alfabético” en la página 91

“Expresiones lógicas” en la página 497

“Expresiones numéricas” en la página 504

“open” en la página 616

“prepare” en la página 630

“replace” en la página 632

“set” en la página 636

“Expresiones de texto” en la página 505

“terminalID” en la página 938

“while” en la página 648

Palabras clave por orden alfabético

Palabra clave	Finalidad
“add” en la página 561	Coloca un registro en un archivo, una cola de mensajes o una base de datos; o coloca un conjunto de registros en una base de datos.
“call” en la página 563	Transfiere el control a otro programa y, opcionalmente, pasa una serie de valores. El control vuelve al llamador cuando el programa llamado finaliza. Si el programa llamado cambia los datos que se han pasado por medio de una variable, cambia también el área de almacenamiento disponible para el llamador.
“case” en la página 566	Marca el inicio de varios conjuntos de sentencias, entre los que sólo se ejecuta uno de los conjuntos como máximo. La sentencia case es equivalente a una sentencia C o Java <i>switch</i> que tengan una interrupción al final de cada cláusula case.
“close” en la página 568	Desconecta una impresora; o cierra el archivo o cola de mensajes asociada con un registro dado; o, en el caso de un registro SQL, cierra el cursor que se abrió mediante una sentencia EGL open o get .
“continue” en la página 570	Presenta un formulario de texto en una aplicación de texto.
“converse” en la página 570	Presenta un formulario de texto en una aplicación de texto.
“delete” en la página 571	Elimina un registro de un archivo o una fila de una base de datos.
“display” en la página 573	Añade un formulario de texto a un almacenamiento intermedio de tiempo de ejecución, pero no presenta datos en la pantalla.
“execute” en la página 574	Permite escribir una o varias sentencias SQL; en particular, sentencias de definición de datos SQL (de tipo CREATE TABLE, por ejemplo) y sentencias de manipulación de datos (de tipo INSERT o UPDATE, por ejemplo).
“exit” en la página 578	Deja el bloque especificado, que por omisión es el bloque que contiene inmediatamente la sentencia exit .
“for” en la página 580	Inicia un bloque de sentencia que se ejecuta en un bucle tantas veces como una prueba de como resultado true.
“forEach” en la página 581	Marca el inicio de un conjunto de sentencias que se ejecutan en un bucle. La primera iteración se produce solamente si un conjunto de resultados especificado está disponible y continúa (en la mayoría de los casos) hasta que se ha procesado la última fila de ese conjunto de resultados.
“forward” en la página 583	Visualiza una página Web con información de variable. Esta sentencia se invoca desde un PageHandler.
“freeSQL” en la página 584	Libera los recursos asociados a una sentencia SQL preparada dinámicamente, cerrando cualquier cursor abierto asociado con esa sentencia SQL.
“get” en la página 585	Recupera un solo registro de archivo o fila de base de datos y proporciona una opción que permite sustituir o suprimir los datos almacenados más tarde en el código. Además, esta sentencia permite recuperar un conjunto de filas de base de datos y sustituir cada fila sucesiva en el siguiente registro SQL de una matriz dinámica. La sentencia get se identifica a veces como get by key value y no es distinta de las sentencias get by position como por ejemplo get next .
“get absolute” en la página 591	Lee una fila especificada numéricamente en un conjunto de resultados de base de datos relacional seleccionado por una sentencia open .

Palabra clave	Finalidad
“get current” en la página 593	Lee la flecha en la que el cursor ya está posicionado en un conjunto de resultados de base de datos seleccionado por una sentencia open .
“get first” en la página 594	Lee la primera fila de un conjunto de resultados de base de datos seleccionado por una sentencia open .
“get last” en la página 595	Lee la última fila de un conjunto de resultados de base de datos seleccionado por una sentencia open .
“get next” en la página 597	Lee el registro siguiente de un archivo o cola de mensajes o la fila siguiente de un conjunto de resultados de base de datos.
“get previous” en la página 602	Lee el registro anterior del archivo asociado a un registro indexado de EGL especificado o lee la fila anterior de un conjunto de resultados de base de datos seleccionado por una sentencia open .
“get relative” en la página 606	Lee una fila especificada numéricamente en un conjunto de resultados de base de datos seleccionado por una sentencia open . La fila se identifica en relación con la posición del cursor en el conjunto de resultados.
“goTo” en la página 608	Hace que el proceso continúe en una etiqueta especificada, que debe estar en la misma función que la sentencia y fuera de un bloque.
“if, else” en la página 608	Marca el inicio de un conjunto de sentencias (si las hay) que sólo se ejecutan si una expresión lógica se resuelve en true. La palabra clave opcional else marca el inicio de un conjunto de sentencias alternativo (si las hay) que sólo se ejecutan si la expresión lógica se resuelve en false. La palabra reservada end marca el cierre de la sentencia if .
“move” en la página 610	Copia datos, byte a byte o por nombre. La última operación copia datos de los elementos nombrados en una estructura a los elementos con el mismo nombre en otra.
“open” en la página 616	Selecciona un conjunto de filas de una base de datos relacional para recuperarlas posteriormente como sentencias get by position como get next . La sentencia open puede operar sobre un cursor o sobre un procedimiento al que se llama.
“prepare” en la página 630	Especifica una sentencia SQL PREPARE, que opcionalmente incluye detalles que sólo se conocen durante la ejecución. La sentencia SQL preparada se ejecuta mediante la ejecución de una sentencia EGL execute o (si la sentencia SQL devuelve un conjunto de resultados) mediante la ejecución de una sentencia EGL open o get .
“print” en la página 632	Añade un formulario de impresión a un almacenamiento intermedio de tiempo de ejecución.
“replace” en la página 632	Coloca un registro cambiado en un archivo o base de datos.
“return” en la página 635	Sale de una función y, opcionalmente, devuelve un valor a la función invocante.
“set” en la página 636	Tiene varios efectos sobre los registros, formularios de texto y elementos.
“show” en la página 646	Presenta un formulario de texto desde un programa principal junto con los demás formularios almacenados en el almacenamiento intermedio mediante la sentencia display ; finaliza el programa actual y, opcionalmente, reenvía los datos de entrada del usuario y los datos de estado del programa actual al programa que maneja la entrada del usuario.
“transfer” en la página 646	Transfiere el control de un programa principal a otro, finaliza el programa que realiza la transferencia y, opcionalmente, pasa un registro cuyos datos se aceptan al registro de entrada del programa receptor. No puede utilizarse una sentencia transfer en un programa llamado.

Palabra clave	Finalidad
“try” en la página 648	Indica que el programa continúa ejecutándose si una sentencia de entrada/salida (E/S), una invocación a función de sistema o una sentencia call provoca un error y se encuentra dentro de la sentencia try . Si se produce una excepción, el proceso se reanuda en la primera sentencia del bloque onException (si la hay), o en la primera sentencia a continuación del final de la sentencia try . No obstante, un error de E/S se maneja solamente si la variable del sistema VGVar.handleHardIOErrors está establecida en 1; de lo contrario, el programa visualiza un mensaje (si es posible) y finaliza.
“while” en la página 648	Marca el inicio de un conjunto de sentencias que se ejecutan en un bucle. La primera ejecución sólo se produce si una expresión lógica se resuelve en true y cada una de las iteraciones subsiguientes depende de la misma prueba. La palabra reservada end marca el cierre de la sentencia while .

Consulta relacionada

“Sentencias EGL” en la página 88

Transferencia de control entre programas

EGL proporciona varias formas de pasar el control de un programa a otro:

- La sentencia **call** proporciona el control a otro programa y, opcionalmente, pasa una serie de valores. El control se devuelve al llamador cuando finaliza el programa llamado. Si el programa llamado cambia los datos que se han pasado en forma de variable, el contenido de la variable se cambia en el llamador.

La llamada no compromete las bases de datos ni otros recursos recuperables, aunque puede producirse un compromiso automático del lado del servidor.

Puede especificar las características de la llamada estableciendo un elemento **callLink** del componente de opciones de enlace. Para obtener información detallada, consulte las secciones *call* y *elemento callLink*. Para obtener información detallada sobre el compromiso del lado del servidor, consulte la sección *luwControl en elemento callLink*.

- La sentencia **transfer** pasa el control de un programa principal a otro, finaliza el programa que realiza la transferencia y, opcionalmente, pasa un registro cuyos datos se aceptan en el *registro de entrada* del programa receptor. No se puede utilizar una sentencia **transfer** en un programa llamado.

El programa puede transferir el control mediante una sentencia del tipo *transferir a transacción* o mediante una sentencia del tipo *transferir a programa*:

- Una sentencia del tipo *transferir a transacción* actúa de la manera siguiente:
 - En un programa que se ejecuta como un programa por lotes principal o de texto principal Java, el comportamiento depende del valor establecido en la opción del descriptor de construcción `synchOnTrxTransfer--`
 - Si el valor de `synchOnTrxTransfer` es YES, la sentencia **transfer** compromete los recursos recuperables, cierra los archivos, cierra los cursores e inicia un programa en la misma unidad de ejecución.
 - Si el valor de `synchOnTrxTransfer` es NO (el valor por omisión), la sentencia **transfer** también inicia un programa en la misma unidad de ejecución, pero no cierra ni compromete los recursos que están disponibles en el programa invocado.
- Una sentencia del tipo *transferir a programa* no compromete ni retrotrae los recursos recuperables, pero cierra los archivos, libera los bloqueos e inicia un programa en la misma unidad de ejecución.

El componente de opciones de enlace no afecta a las características de ninguno de estos tipos de transferencia.

En un PageHandler, una transferencia no es válida.

Para obtener información detallada, consulte las secciones *transfer*.

- La función de sistema **sysLib.startTransaction** inicia una unidad de ejecución de forma asíncrona. La operación no finaliza el programa que realiza la transferencia y no afecta a las bases de datos, archivos y bloqueos del programa que realiza la transferencia. Tiene la opción de pasar datos al *registro de entrada*, que es un área del programa receptor.

Si el programa invoca **sysLib.startTransaction**, debe generar el programa con un componente de opciones de enlace, el elemento **asynchLink**. Para obtener información detallada, consulte las secciones *sysLib.startTransaction* y *elemento asynchLink*.

- La sentencia EGL **show** finaliza el programa principal actual de una aplicación de texto y muestra los datos al usuario mediante un formulario. Después de que el usuario haya enviado el formulario, la sentencia **show** opcionalmente reenvía el control a un segundo programa principal, el cual recibe los datos recibidos del usuario así como los datos que se han pasado sin ningún cambio desde el programa originador.

La sentencia **show** se ve afectada por los valores establecidos en el componente de opciones de enlace, el elemento **transferLink**

Para obtener información detallada, consulte la sección *show*.

- Finalmente, la sentencia **forward** se invoca desde un PageHandler o un programa. La sentencia actúa de la manera siguiente:
 1. Compromete los recursos recuperables, cierra los archivos y libera los bloqueos
 2. Reenvía el control
 3. Finaliza el código

El destino en este caso es otro programa o página Web. Para obtener información detallada, consulte la sección *forward*.

Consulta relacionada

“Elemento asynchLink” en la página 377

“call” en la página 563

“Elemento callLink” en la página 407

“forward” en la página 583

“Propiedad luvControl del elemento callLink” en la página 414

“show” en la página 646

“startTransaction()” en la página 911

“transfer” en la página 646

Manejo de excepciones

Se puede producir un error cuando un programa generado por EGL actúa de la manera siguiente:

- Accede a un archivo, cola o base de datos
- Llama a otro programa
- Invoca una función
- Realiza una asignación, comparación o cálculo

Bloques try

Un *bloque try* EGL es una serie de cero a muchas sentencias EGL que están dentro de los delimitadores **try** y **end**. A continuación se ofrece un ejemplo:

```
if (userRequest = "A")
  try
    add record1;
  onException
    myErrorHandler(12);
  end
end
```

En general, un bloque try permite que el programa se siga procesando aunque se produzca un error.

El bloque try puede incluir una cláusula *onException*, como se ha mostrado en el ejemplo anterior. Esta cláusula se invoca si falla una de las sentencias anteriores del bloque try; pero en ausencia de una cláusula *onException*, un error en un bloque try hace que se invoque la primera sentencia que sigue inmediatamente al bloque try.

Excepciones del sistema de EGL

EGL proporciona una serie de excepciones del sistema para indicar la naturaleza específica de un problema de tiempo de ejecución. Cada una de estas excepciones es un diccionario desde el que puede recuperarse información, pero la recuperación siempre se hace por medio de la variable del sistema **SysLib.currentException** (también un diccionario), que permite acceder a la excepción mostrada más recientemente en la unidad de ejecución.

Un campo de cualquier excepción es **code**, que es una serie que identifica la excepción. Puede determinar la excepción actual probando ese campo en la lógica de la forma siguiente:

```
if (userRequest = "A")
  try
    add record1;
  onException
    case (SysLib.currentException.code)
      when (FileIOException)
        myErrorHandler(12);
      otherwise
        myErrorHandler(15);
    end
  end
end
```

En este caso, `FileIOException` es una constante que es equivalente al valor de serie "com.ibm.egl.FileIOException". La constante de excepción de EGL siempre es equivalente al último calificador de una serie que empieza por "com.ibm.egl".

Se recomienda encarecidamente que acceda a los campos de excepción solo en un bloque *onException*. La unidad de ejecución termina si el código accede a **SysLib.currentException** cuando no se ha producido ninguna excepción.

El ejemplo siguiente accede al campo `sqlcode` en la excepción `SQLException`:

```
if (userRequest = "A")
  try
    add record01;
  onException
    case (SysLib.currentException.code)
```

```

        when ("com.ibm.egl.SQLErrorException")
            if (SysLib.currentException.sqlcode == -270)
                myErrorHandler(16);
            else
                myErrorHandler(20);
            end
        otherwise
            myErrorHandler(15);
        end
    end
end
end

```

Para conocer más detalles sobre las excepciones del sistema, consulte *Excepciones del sistema EGL*.

Límites de los bloques try

Los detalles anteriores sobre los bloques try deben estar calificados. En primer lugar, un bloque try sólo afecta al proceso de errores en los siguientes tipos de sentencias EGL:

- Una sentencia de E/S
- Una función de sistema
- Una sentencia call

El proceso de los desbordamientos numéricos no se ve afectado por la presencia de un bloque try. Para obtener información detallada sobre estos tipos de error, consulte la sección *VGVar.handleOverflow*.

En segundo lugar, un bloque try no tiene ningún efecto sobre los errores de una función de usuario (o programa) que se invoca desde dentro del bloque try. En el siguiente ejemplo, si una sentencia falla en la función myABC, el programa finaliza inmediatamente con un mensaje de error a menos que la propia función myABC maneje el error:

```

    if (userRequest = "B")
        try
            myVariable = myABC();
        onException
            myErrorHandler(12);
        end
    end
end

```

En tercer lugar, el programa finaliza inmediatamente y con un mensaje de error en los siguientes casos:

- Un error de un tipo que está incluido específicamente en un bloque try se produce fuera de un bloque try; o bien
- Se aplica uno de los siguientes casos, incluso en un bloque try:
 - Se produce una anomalía en la invocación o retorno de una función escrita por el usuario;
 - Se asigna un carácter no numérico a una variable numérica; o bien
 - La variable de sistema **VGVar.handleHardIOErrors** se establece en cero en lugar de en uno cuando una sentencia de E/S de archivo finaliza con un *error grave* (como se describe más adelante).

Los casos siguientes también son de interés:

- Si un valor se divide por cero, un programa Java maneja la situación como un desbordamiento numérico

- Una unidad de ejecución Java finaliza si se asigna un carácter no numérico a una variable numérica del descriptor de construcción

Nota: Para dar soporte a la migración de programas escritos en VisualAge Generator y EGL 5.0, la variable `VGVar.handleSysLibraryErrors` (anteriormente llamada `ezereply`) permite procesar algunos errores que se producen fuera de un bloque `try`. Evite utilizar esa variable, que sólo está disponible si trabaja en modalidad de compatibilidad de VisualAge Generator.

Variables de sistema relacionadas con errores

EGL proporciona variables de sistema relacionadas con errores que se establecen en un bloque `try` como respuesta a eventos satisfactorios o como respuesta a errores de no terminación. Los valores de dichas variables están disponibles en el bloque `try` y en el código que se ejecuta después del bloque `try` y, en la mayoría de los casos, los valores se restauran después de una sentencia `converse`, si existe.

El entorno de ejecución EGL no cambia el valor de ninguna variable relacionada con errores cuando las sentencias se ejecutan fuera de un bloque `try`. Sin embargo, el programa puede asignar un valor a una variable relacionada con errores fuera de un bloque `try`.

En algunas situaciones se asigna un valor a la variable de sistema `sysVar.exceptionCode` y en todas estas situaciones también se establece una o más variables adicionales, en función de la naturaleza de la interacción del programa con el entorno de ejecución:

- Se asignan valores a las variables de sistema `sysVar.exceptionCode` y `sysVar.errorCode` después de que se ejecute alguno de los siguientes tipos de sentencias en un bloque `try`:
 - Una sentencia `call`
 - Una sentencia de E/S que actúa sobre un archivo indexado, MQ, relativo o serie
 - Una invocación de casi cualquier función de sistema
- Se asignan valores a las variables del sistema `sysVar.exceptionCode`, `sysVar.errorCode`, `VGVar.mqConditionCode` y `sysVar.mqReasonCode` después de que una sentencia de E/S de un bloque `try` actúe sobre un registro MQ
- La variable de sistema `sysVar.exceptionCode` recibe un valor una vez que se ha accedido a una base de datos relacional desde una sentencia de un bloque `try`. También se asignan valores a variables del área de comunicación SQL (SQLCA); para obtener detalles, consulte `sysVar.sqlca`.

Si se produce un error de no terminación en un bloque `try`, el valor de `sysVar.exceptionCode` es equivalente al componente numérico del mensaje de error EGL que se presentaría al usuario si el error se produjese fuera del bloque `try`. Sin embargo, el sistema de ejecución proporciona los valores de las variables específicas de la situación, como por ejemplo `sysVar.errorCode` y `VGVar.mqConditionCode`. En ausencia de un error, el valor de `sysVar.exceptionCode` y de al menos una de las variables específicas de la situación es el mismo: una serie de ocho ceros.

Se asigna un código de error a `sysVar.exceptionCode` y `sysVar.errorCode` en el caso de un desbordamiento numérico de no terminación, como se describe en la sección `VGVar.handleOverflow`; pero un cálculo aritmético satisfactorio no afecta a ninguna de las variables de sistema relacionadas con errores.

Las variables de sistema relacionadas con errores tampoco se ven afectadas por la invocación de una función que no sea una función de sistema, y **sysVar.errorCode** (la variable afectada por la mayoría de funciones de sistema) no se ve afectada por los errores en las siguientes funciones:

- **sysLib.calculateChkDigitMod10**
- **sysLib.calculateChkDigitMod11**
- **strLib.concatenate**
- **strLib.concatenateWithSeparator**
- **VGLib.connectionService**
- **sysLib.connect**
- **sysLib.convert**
- **sysLib.disconnect**
- **sysLib.disconnectAll**
- **sysLib.purge**
- **sysLib.queryCurrentDatabase**
- **strLib.setBlankTerminator**
- **sysLib.setCurrentDatabase**
- **strLib.strLen**
- **sysLib.verifyChkDigitMod10**
- **sysLib.verifyChkDigitMod11**
- **sysLib.wait**

Cuando se asigna un valor de error a **sysVar.exceptionCode**, a la variable de sistema **sysVar.exceptionMsg** se le asigna el texto del mensaje de error EGL relacionado y a la variable de sistema **sysVar.exceptionMsgCount** se le asigna el número de bytes del mensaje de error, excluidos los nulos y blancos finales. Cuando se asigna una serie de ocho ceros a **sysVar.exceptionCode**, se asignan blancos a **sysVar.exceptionMsg** y **sysVar.exceptionMsgCount** se establece en cero.

Sentencias de E/S

En relación con las sentencias de E/S, un error puede ser grave o leve:

- Un error leve puede ser cualquiera de los siguientes:
 - No se ha encontrado ningún registro durante una operación de E/S en una tabla de base de datos
 - Se produce uno de los siguientes problemas en una operación de E/S en un archivo indexado, relativo o serie:
 - Registro duplicado (cuando el almacén de datos externo permite insertar un duplicado)
 - No se ha encontrado ningún registro
 - Fin de archivo
- Un error grave puede ser cualquier otro problema, como por ejemplo:
 - Registro duplicado (cuando el almacén de datos externo prohíbe insertar un duplicado)
 - No se ha encontrado un archivo
 - Los enlaces de comunicación no están disponibles durante el acceso remoto de un conjunto de datos

Si la sentencia que causa el error leve está en un bloque try, se aplican las sentencias siguientes:

- Por omisión, EGL continúa ejecutándose sin pasar el control al bloque onException

- Si desea pasar el control al bloque `OnException`, establezca la propiedad `throwNrfEofExceptions` en *yes* en un programa, `pageHandler` o biblioteca

Si se produce un error grave de E/S en un bloque `try`, la consecuencia depende del valor de una variable de sistema relacionada con errores:

- Durante el acceso de un archivo, base de datos relacional o cola de mensajes `MQSeries`, se aplican las siguientes normas:
 - Si `VGVar.handleHardIOErrors` se establece en 1, el programa continúa ejecutándose
 - Si `VGVar.handleHardIOErrors` se establece en 0, el programa presenta un mensaje de error, si es posible, y finaliza

El valor por omisión de esa variable es dependiente del valor de la propiedad `handleHardIOErrors`, que está disponible en componentes de lógica generable como programas, bibliotecas y `pageHandlers`. El valor por omisión para la propiedad es *yes* que establece el valor inicial de la variable `VGVar.handleHardIOErrors` en 1.

Si se produce un error grave o leve de E/S fuera de un bloque `try`, el programa generado presenta un mensaje de error, si es posible, y finaliza.

Si accede a DB2 directamente (no a través de JDBC), el `sqlcode` para un error grave es 304, 802 o menor de 0.

Identificación de errores

Puede determinar qué tipo de error se ha producido en un bloque `try` incluyendo una sentencia `case` o `if` dentro o fuera del bloque `try`, y en esta sentencia puede probar el valor de diversas variables de sistema. Sin embargo, si responde a un error de E/S y si la sentencia utiliza un registro `EGL`, se recomienda utilizar una expresión lógica básica. Están disponibles dos formatos de la expresión:

nombreRegistro **es** *valorErrorES*

nombreRegistro **no es** *valorErrorES*

nombreRegistro

Nombre del registro utilizado en la operación de E/S

valorErrorES

Uno de varios valores de error de E/S que son constantes en diferentes sistemas de gestión de bases de datos

Si no utiliza las expresiones lógicas con valores de error de E/S y luego cambia los sistemas de gestión de base de datos, es posible que necesite modificar y volver a generar el programa. En particular, es aconsejable utilizar los valores de error de E/S para comprobar los errores en lugar del valor de `sysVar.sqlcode` o `sysVar.sqlState`. Dichos valores dependen de la implementación de base de datos subyacente.

Conceptos relacionados

“Compatibilidad con VisualAge Generator” en la página 439

“Diccionario” en la página 82

Consulta relacionada

“Código de error de ejecución de Java EGL” en la página 959

“Sentencias EGL” en la página 88

“Valores de error de E/S” en la página 536

“Expresiones lógicas” en la página 497

"errorCode" en la página 931
"overflowIndicator" en la página 932
"sqlca" en la página 935
"sqlcode" en la página 936
"sqlState" en la página 936
"handleSysLibraryErrors" en la página 947
"handleHardIOErrors" en la página 946
"handleOverflow" en la página 946
"mqConditionCode" en la página 948

Migrar código EGL a EGL 6.0 iFix

La herramienta de migración de EGL V6.0 convierte el código fuente de EGL V5.1.2 y V6.0 para que se ajuste a EGL V6.0 iFix. Esta herramienta puede utilizarse sobre todo un proyecto, un solo archivo o una selección de archivos. La ejecución de la herramienta sobre un paquete o una carpeta convierte todos los archivos fuente de EGL en ese paquete o carpeta. Para obtener más información acerca del código cambiado por la herramienta de migración, consulte el apartado *Migración de EGL a EGL*.

Nota: No utilice la herramienta de migración sobre el código que ya se haya actualizado a EGL V6.0 iFix. Si lo hace, pueden producirse errores en el código.

Para migrar código EGL a EGL V6.0 iFix, haga lo siguiente:

1. En el entorno de trabajo, pulse **Ventana > Preferencias**.
2. En el lado izquierdo de la ventana Preferencias, expanda **Entorno de trabajo** y pulse **Posibilidades**.
3. En la lista de posibilidades, expanda **Desarrollador EGL**.
4. Marque el recuadro de selección de la posibilidad llamada **Migración de EGL V6.0**.
5. Pulse en **Aceptar**.
6. De nuevo, pulse **Ventana > Preferencias**.
7. En el lado izquierdo de la ventana Preferencias, expanda **EGL** y pulse **Preferencias de migración de EGL V6.0**.
8. Establezca las preferencias para la herramienta de migración EGL V6.0. Para obtener más información acerca de las preferencias, consulte el apartado *Establecer preferencias de migración de EGL a EGL*.
9. En la vista Explorador de proyectos o en la vista Navegador, seleccione los proyectos EGL, paquetes, carpetas o archivos que desee migrar. Puede seleccionar cualquier número de recursos EGL para la migración. Para seleccionar más de un recurso a la vez, mantenga pulsada la tecla CTRL al pulsar los recursos.
10. Pulse el botón derecho del ratón sobre un recurso seleccionado y pulse **Migración de EGL V6.0 > Migrar** en el menú emergente.
11. En el código, busque lugares que no se ajusten al iFix de EGL V6.0.

La herramienta de migración convierte los archivos fuente EGL seleccionados para que se ajusten a EGL V6.0 iFix. Para revisar los cambios realizados por la herramienta en el código fuente, haga lo siguiente:

1. En la vista Explorador de proyectos o en la vista Navegador, pulse con el botón derecho del ratón sobre un archivo fuente EGL que se haya migrado y pulse **Comparar con > Historial local** en el menú emergente.
2. Examine las diferencias entre el archivo del entorno de trabajo y la versión anterior.
3. Cuando haya terminado de revisar los cambios, pulse **Aceptar**.

Conceptos relacionados

“Migración de EGL a EGL”

“Establecer las preferencias de migración de EGL a EGL” en la página 110

Tareas relacionadas

“Habilitar posibilidades de EGL” en la página 122

Migración de EGL a EGL

La herramienta de migración de EGL V6.0 convierte el código fuente de EGL V5.1.2 y V6.0 para que se ajuste a EGL V6.0 iFix. Esta herramienta puede utilizarse sobre todo un proyecto, un solo archivo o una selección de archivos. La ejecución de la herramienta sobre un paquete o una carpeta convierte todos los archivos fuente de EGL en ese paquete o carpeta. Para obtener instrucciones sobre cómo utilizar la herramienta de migración, consulte la sección *Migrar código EGL a EGL 6.0 iFix*.

La herramienta de migración puede añadir comentarios a cada archivo si cambia y también puede añadir comentarios al archivo de anotaciones del proyecto. Para cambiar estas opciones, consulte *Preferencias de migración de EGL a EGL*.

La herramienta de migración realiza los cambios siguientes en el código fuente de EGL para que se ajuste a EGL V6.0 iFix:

- La herramienta de migración cambia la forma en la que se especifican las propiedades. Para obtener información acerca de los cambios en las propiedades, consulte la sección *Cambios en las propiedades durante la migración de EGL a EGL*.
- La herramienta de migración busca variables y nombres de componentes que entren en conflicto con palabras reservadas. La herramienta de migración cambia esos nombres de variables y componentes añadiendo un prefijo o un sufijo según se haya definido en las preferencias de migración de EGL a EGL. Por omisión, la herramienta añade el sufijo `_EGL` a cualquier nombre que sea ahora una palabra reservada. La herramienta de migración no redenomina objetos de la sentencia `CALL` ni actualiza referencias en los archivos de componente de construcción de EGL. Consulte la sección *Palabras reservadas de EGL*. A continuación se proporciona un ejemplo de código antes y después de utilizar la herramienta de migración.

Antes de la migración:

```
Library Handler
  boolean Bin(4);
End
```

Después de la migración:

```
Library Handler_EGL
  boolean_EGL Bin(4);
End
```

- La herramienta de migración sustituye el signo de igualdad (`=`) por el doble signo de igualdad (`==`) cuando se utiliza como un operador de comparación. No cambia el signo de igualdad cuando se utiliza como un operador de asignación.

Antes de la migración:

```
Function test(param int)
  a int;
  If(param = 3)
    a = 0;
  End
End
```

Después de la migración:

```

Function test(param int)
  a int;
  If(param == 3)
    a = 0;
  End
End

```

- La herramienta de migración añade números de nivel a los registros que no tienen números de nivel.

Antes de la migración:

```

Record MyRecord
  item1 int;
  item2 int;
End

```

Después de la migración:

```

Record MyRecord
  10 item1 int;
  10 item2 int;
End

```

- La herramienta de migración cambia la sintaxis de declaración de constantes.

Antes de la migración:

```
intConst 3;
```

Después de la migración:

```
const intConst int = 3;
```

- La herramienta de migración cambia nombres de variables y funciones que se han redenido o movido a distintas bibliotecas. Este cambio afecta a las variables y las funciones de las bibliotecas SysLib y SysVar.

Antes de la migración:

```

SysLib.java();
clearRequestAttr();

```

Después de la migración:

```

JavaLib.invoke();
J2EELib.clearRequestAttr();

```

A continuación se proporciona una lista de nombres de variables y funciones cambiados de las bibliotecas SysLib y SysVar:

Tabla 1. Nombres de variables y funciones cambiados de las bibliotecas SysLib y SysVar

Antes de la migración	Después de la migración
SysLib.dateValue	DateTimeLib.dateValue
SysLib.extendTimestampValue	DateTimeLib.extend
SysLib.formatDate	StrLib.formatDate
SysLib.formatTime	StrLib.formatTime
SysLib.formatTimestamp	StrLib.formatTimestamp
SysLib.intervalValue	DateTimeLib.intervalValue
SysLib.timeValue	DateTimeLib.timeValue
SysLib.timeStampValue	DateTimeLib.timestampValue
SysLib.java	JavaLib.invoke
SysLib.javaGetField	JavaLib.getField
SysLib.javaIsNull	JavaLib.isNull
SysLib.javaIsObjID	JavaLib.isObjID
SysLib.javaRemove	JavaLib.remove

Tabla 1. Nombres de variables y funciones cambiados de las bibliotecas SysLib y SysVar (continuación)

Antes de la migración	Después de la migración
SysLib.javaRemoveAll	JavaLib.removeAll
SysLib.javaSetField	JavaLib.setField
SysLib.javaStore	JavaLib.store
SysLib.javaStoreCopy	JavaLib.storeCopy
SysLib.javaStoreField	JavaLib.storeField
SysLib.javaStoreNew	JavaLib.storeNew
SysLib.javaType	JavaLib.qualifiedTypeName
SysLib.clearRequestAttr	J2EELib.clearRequestAttr
SysLib.clearSessionAttr	J2EELib.clearSessionAttr
SysLib.getRequestAttr	J2EELib.getRequestAttr
SysLib.getSessionAttr	J2EELib.getSessionAttr
SysLib.setRequestAttr	J2EELib.setRequestAttr
SysLib.setSessionAttr	J2EELib.setSessionAttr
SysLib.displayMsgNum	ConverseLib.displayMsgNum
SysLib.clearScreen	ConverseLib.clearScreen
SysLib.fieldInputLength	ConverseLib.fieldInputLength
SysLib.pageEject	ConverseLib.pageEject
SysLib.validationFailed	ConverseLib.validationFailed
SysLib.getVAGSysType	VGLib.getVAGSysType
SysLib.connectionService	VGLib.connectionService
SysVar.systemGregorianCalendar	VGVar.systemGregorianCalendar
SysVar.systemJulianDate	VGVar.systemJulianDate
SysVar.currentDate	VGVar.currentGregorianCalendar
SysVar.currentFormattedDate	VGVar.currentFormattedGregorianCalendar
SysVar.currentFormattedJulianDate	VGVar.currentFormattedJulianDate
SysVar.currentFormattedTime	VGVar.currentFormattedTime
SysVar.currentJulianDate	VGVar.currentJulianDate
SysVar.currentShortDate	VGVar.currentShortGregorianCalendar
SysVar.currentShortJulianDate	VGVar.currentShortJulianDate
SysVar.currentTime	DateTimeLib.currentTime
SysVar.currentTimeStamp	DateTimeLib.currentTimeStamp
SysVar.handleHardIOErrors	VGVar.handleHardIOErrors
SysVar.handleSysLibErrors	VGVar.handleSysLibraryErrors
SysVar.handleOverflow	VGVar.handleOverflow
SysVar.mqConditionCode	VGVar.mqConditionCode
SysVar.sqlerrd	VGVar.sqlerrd
SysVar.sqlerrmc	VGVar.sqlerrmc
SysVar.sqlIsolationLevel	VGVar.sqlIsolationLevel
SysVar.sqlWarn	VGVar.sqlWarn

Tabla 1. Nombres de variables y funciones cambiados de las bibliotecas SysLib y SysVar (continuación)

Antes de la migración	Después de la migración
SysVar.commitOnConverse	ConverseVar.commitOnConverse
SysVar.eventKey	ConverseVar.eventKey
SysVar.printerAssociation	ConverseVar.printerAssociation
SysVar.segmentedMode	ConverseVar.segmentedMode
SysVar.validationMsgNum	ConverseVar.validationMsgNum

- La herramienta de migración cambia la forma en que se especifican las fechas, horas e indicaciones de la hora. A continuación se proporcionan algunos ejemplos:

Tabla 2. Cambios en fechas, horas e indicaciones de la hora

Antes de la migración	Después de la migración
dateFormat = "aa/mm/dd"	dateFormat = "aa/MM/dd"
dateFormat = "AAAA/MM/DD"	dateFormat = "aaaa/MM/dd"
dateFormat = "AAAA/DDD"	dateFormat = "aaaa/DDD"
timeFormat = "hh:mm:ss"	timeFormat = "HH:mm:ss"

- La herramienta de migración establece la propiedad HandleHardIOErrors en no para todas las bibliotecas, programas y PageHandlers migrados para los que no se especifica esa propiedad.

Tareas relacionadas

"Migrar código EGL a EGL 6.0 iFix" en la página 101

Conceptos relacionados

"Establecer las preferencias de migración de EGL a EGL" en la página 110

"Cambios en las propiedades durante la migración de EGL a EGL"

Consulta relacionada

"Palabras reservadas EGL" en la página 486

Cambios en las propiedades durante la migración de EGL a EGL

La herramienta de migración cambia significativamente la forma en que se especifican las propiedades. A continuación se proporciona un resumen de estos cambios:

- La herramienta de migración redenomina propiedades cuyos nombres hayan cambiado en EGL V6.0 iFix. A continuación se proporciona una lista de las propiedades red denominadas:

Tabla 3. Propiedades red denominadas

Antes de la migración	Después de la migración
action	actionFunction
boolean	isBoolean
getOptions	getOptionsRecord
msgDescriptor	msgDescriptorRecord
onPageLoad	onPageLoadFunction

Tabla 3. Propiedades red denominadas (continuación)

Antes de la migración	Después de la migración
openOptions	openOptionsRecord
putOptions	putOptionsRecord
queueDescriptor	queueDescriptorRecord
range	validValues
rangeMsgKey	validValuesMsgKey
selectFromList	selectFromListItem
sqlVar	sqlVariableLen
validator	validatorFunction
validatorMsgKey	validatorFunctionMsgKey
validatorTable	validatorDataTable
validatorTableMsgKey	validatorDataTableMsgKey

- La herramienta de migración añade comillas dobles a valores de propiedad utilizados como literales de serie.

Antes de la migración:

```
{ alias = prog }
```

Después de la migración:

```
{ alias = "prog" }
```

Se ven afectadas las propiedades siguientes:

- alias
 - column
 - currency
 - displayName
 - fileName
 - fillCharacter
 - help
 - helpKey
 - inputRequiredMsgKey
 - minimumInputMsgKey
 - msgResource
 - msgTablePrefix
 - pattern
 - queueName
 - rangeMsgKey
 - tableNames
 - title
 - typeChkMsgKey
 - validatorMsgKey
 - validatorTableMsgKey
 - value
 - view
- La herramienta de migración sustituye paréntesis por corchetes rectangulares al especificar literales de matriz como valores para propiedades.

- formSize
 - keyItems
 - outline
 - pageSize
 - position
 - range
 - screenSize
 - screenSizes
 - tableNames
 - tableNameVariables
 - validationBypassFunctions
 - validationBypassKeys
- Para propiedades que toman literales de matriz, la herramienta de migración pone literales de matriz de un solo elemento entre corchetes para especificar que una matriz con un solo elemento sigue siendo una matriz. La herramienta de migración utiliza corchetes dobles para las propiedades que toman matrices de matrices.

Antes de la migración:

```
{ keyItems = var, screenSizes = (24, 80), range = (1, 9) }
```

Después de la migración:

```
{ keyItems = ["var"], screenSizes = [[24, 80]], range = [[1, 9]] }
```

- La herramienta de migración utiliza la palabra clave **this** en lugar de un nombre de variable cuando altera temporalmente las propiedades para un elemento específico en una matriz.

Antes de la migración:

```
Form myForm type TextForm
  fieldArray char(10)[5] { fieldArray[1] {color = red } };
end
```

Después de la migración:

```
Form myForm type TextForm
  fieldArray char(10)[5] { this[1] {color = red } };
end
```

- La herramienta de migración cambia referencias a componentes, funciones y campos añadiendo comillas y corchetes donde corresponda.

Antes de la migración:

```
{ keyItems = (item1, item2) }
```

Después de la migración:

```
{ keyItems = ["item1", "item2"] }
```

Las propiedades siguientes se ven afectadas por la herramienta de migración de esta forma:

- action
- commandValueItem
- getOptions
- helpForm
- inputForm
- inputPageRecord
- inputRecord
- keyItem

- keyItems
 - lengthItem
 - msgDescriptorRecord
 - msgField
 - numElementsItem
 - onPageLoadFunction
 - openOptionsRecord
 - putOptionsRecord
 - queueDescriptorRecord
 - redefines
 - selectFromListItem
 - tableNameVariables
 - validationBypassFunctions
 - validatorFunction
 - validatorDataTable
- La herramienta de migración asigna el valor por omisión *yes* a cualesquiera propiedades booleanas que se hayan especificado pero a las que no se haya asignado un valor.

Antes de la migración:

```
{ isReadOnly }
```

Después de la migración:

```
{ isReadOnly = yes }
```

Las propiedades siguientes se ven afectadas por la herramienta de migración de esta forma:

- addSpaceForSOSI
- allowUnqualifiedItemReferences
- boolean
- bypassValidation
- containerContextDependent
- currency
- cursor
- deleteAfterUse
- detectable
- fill
- helpGroup
- includeMsgInTransaction
- includeReferencedFunctions
- initialized
- inputRequired
- isDecimalDigit
- isHexDigit
- isNullable
- isReadOnly
- lowerCase
- masked
- modified

- needsSOSI
 - newWindow
 - numericSeparator
 - openQueueExclusive
 - pfKeyEquate
 - resident
 - runValidatorFromProgram
 - segmented
 - shared
 - sqlVar
 - upperCase
 - wordWrap
 - zeroFormat
- La herramienta de migración divide la propiedad **currency** en dos propiedades: **currency** y **currencySymbol**. En la tabla siguiente se proporcionan algunos de ejemplos del cambio de la propiedad **currency** por parte de la herramienta de migración.

Tabla 4. Cambios en la propiedad **currency**

Antes de la migración	Después de la migración
{ currency = yes }	{ currency = yes }
{ currency = no }	{ currency = no }
{ currency = "usd" }	{ currency = yes, currencySymbol = "usd" }

- La herramienta de migración cambia los valores de las propiedades **dateFormat** y **timeFormat** para que sean sensibles a las mayúsculas/minúsculas. Para obtener más información, consulte *Especificadores de formato de Fecha, hora e indicación de la hora*.
- Si el recuadro de selección **Añadir calificadores a valores de propiedad de enumeración** está marcado en el menú de preferencias, la herramienta de migración añade el tipo de valor al valor de la propiedad.

Antes de la migración:

```
color = red
outline = box
```

Después de la migración:

```
color = ColorKind.red
outline = OutlineKind.box
```

Este cambio afecta a las propiedades siguientes:

- align
- color
- deviceType
- displayUse
- highlight
- indexOrientation
- intensity
- outline
- protect

- selectType
- sign
- La herramienta de migración cambia los valores de la propiedad **tableNames** para que sean una matriz de matrices de series. Cada matriz de series debe tener uno o dos elementos. El primer elemento es el nombre de la tabla y el segundo, si lo hay, la etiqueta de la tabla. En la tabla siguiente se proporcionan algunos de ejemplos del cambio de la propiedad **tableNames** por parte de la herramienta de migración.

Tabla 5. Cambios en la propiedad **tableNames**

Antes de la migración	Después de la migración
{ tableNames = (table1, table2) }	{ tableNames = [{"table1"}, {"table2"}] }
{ tableNames = (table1 t1, table2) }	{ tableNames = [{"table1", "t1"}, {"table2"}] }
{ tableNames = (table1 t1, table2 t2) }	{ tableNames = [{"table1", "t1"}, {"table2", "t2"}] }

- La herramienta de migración cambia los valores de la propiedad **tableNameVariables** de la misma forma que cambia los valores de la propiedad **tableNames**.
- La herramienta de migración cambia los valores de la propiedad **defaultSelectCondition** para que sean de tipo `sqlCondition`.

Antes de la migración:

```
{ defaultSelectCondition =
  #sql{
    hostVar02 = 4
  }
}
```

Después de la migración:

```
{ defaultSelectCondition =
  #sqlCondition{ // sin espacio entre #sqlCondition y la llave
    hostVar02 = 4
  }
}
```

- La herramienta de migración sustituye el valor NULL de **fillCharacter** por el valor de serie vacía "".

Tareas relacionadas

“Migrar código EGL a EGL 6.0 iFix” en la página 101

Conceptos relacionados

“Migración de EGL a EGL” en la página 102

“Establecer las preferencias de migración de EGL a EGL”

“Especificadores de fecha, hora e indicación de la hora” en la página 45

Establecer las preferencias de migración de EGL a EGL

Puede establecer las preferencias que controlan cómo la herramienta de migración de EGL V6.0 convierte el código fuente de EGL. Para obtener más información acerca de la herramienta de migración, consulte el apartado *Migración de EGL a EGL*. Establezca las preferencias de la herramienta de migración de la manera siguiente:

1. Pulse en **Ventana > Preferencias**.
2. Expanda **EGL**.
3. Pulse **Preferencias de migración de EGL V6.0**.

Nota: Si no encuentra **Preferencias de migración de EGL V6.0**, habilite la posibilidad Migración de EGL V6.0. Consulte el apartado *Habilitar posibilidades de EGL*.

4. Elija cómo resolver un conflicto de nombres con una palabra reservada nueva pulsando un botón de selección:
 - **Añadir prefijo** establece que la herramienta de migración añada un prefijo a las palabras del código fuente que ahora son palabras reservadas. En el recuadro de texto situado junto a este botón de selección, teclee el prefijo que la herramienta de migración debe añadir a la palabra cambiada.
 - **Añadir sufijo** establece que la herramienta de migración añada un sufijo a las palabras del código fuente que ahora son palabras reservadas. En el recuadro de texto situado junto a este botón de selección, teclee el sufijo que la herramienta de migración debe añadir a la palabra cambiada.
5. Para añadir un calificador a los valores de las propiedades que tengan una lista finita de valores posibles, marque el recuadro de sesión **Añadir calificadores a los valores de propiedad de enumeración**. Si se marca este recuadro de selección, la herramienta de migración añadirá el tipo de valor al nombre de valor.
6. Para añadir comentarios a los archivos cambiados por la herramienta de migración, elija una opción bajo **Nivel de anotación**.
 - **Añadir comentarios a todos los archivos procesados por la herramienta de migración** establece que la herramienta de migración añada un comentario a cada archivo que procese, incluso aunque no haga cambios en ese archivo.
 - **Añadir comentarios a todos los archivos cambiados por la herramienta de migración** establece que la herramienta de migración añada un comentario solo a los archivos que cambia.
 - **No añadir comentarios a los archivos** establece que la herramienta de migración no añada comentarios a los archivos que procesa.
 - **Añadir al principio del archivo** establece que la herramienta de migración añada comentarios al principio de los archivos.
 - **Añadir al final del archivo** establece que la herramienta de migración añada comentarios al final de los archivos.
7. Para escribir una lista de los archivos procesados al archivo llamado V60MigrationLog.txt, marque el recuadro de selección **Añadir resultados de la migración al archivo de anotaciones por proyecto**.
8. Pulse en **Aplicar**.
9. Pulse en **Aceptar**.

Tareas relacionadas

“Habilitar posibilidades de EGL” en la página 122

“Migrar código EGL a EGL 6.0 iFix” en la página 101

Conceptos relacionados

“Migración de EGL a EGL” en la página 102

Configurar el entorno

Establecer preferencias de EGL

Establezca las preferencias básicas de EGL como se indica a continuación:

1. Pulse en **Ventana > Preferencias**.
2. Cuando se visualice una lista, pulse en **EGL** para visualizar la pantalla de EGL.
3. Seleccione o deseleccione el recuadro de selección para **Compatibilidad con VisualAge Generator**. Su elección afectará a las opciones disponibles en el momento del desarrollo, tal como se describe en *Compatibilidad con VisualAge Generator*.
4. En el cuadro de lista **Codificación**, seleccione el juego de codificación de caracteres que se utilizará al crear nuevos archivos de construcción de EGL (.eglbld). El valor establecido no tiene efecto alguno sobre los archivos de construcción existentes. El valor por omisión es UTF-8.
5. En el cuadro de texto **ID de usuario**, especifique el ID de usuario para acceder a la máquina de construcción remota, si la hay. La opción del descriptor de construcción **destUserID** tiene prioridad y tanto esa opción como el valor de preferencia tienen prioridad sobre la opción original del descriptor de construcción **destUserID**.
6. En el cuadro de texto **Contraseña**, especifique la contraseña para acceder a la máquina de construcción remota, si la hay. La opción del descriptor de construcción **destPassword** tiene prioridad y tanto esa opción como el valor de preferencia tienen prioridad sobre la opción original del descriptor de construcción **destPassword**.
7. Pulse en **Aplicar**.

Para establecer otras preferencias de EGL, consulte la lista de tareas relacionadas en la parte inferior de esta página. Cuando haya terminado de establecer preferencias, pulse en **Aceptar**.

Conceptos relacionados

“Construcción” en la página 326

“Compatibilidad con VisualAge Generator” en la página 439

Tareas relacionadas

“Establecer los descriptores de construcción por omisión” en la página 116

“Establecer preferencias para el depurador de EGL” en la página 114

“Establecer preferencias para estilos de fuente” en la página 117

“Establecer preferencias para conexiones a bases de datos SQL” en la página 119

“Establecer preferencias para la recuperación de SQL” en la página 121

Establecer las preferencias del texto

Para cambiar la visualización del texto en el editor de EGL, haga lo siguiente:

1. Pulse en **Ventana > Preferencias**.
2. Cuando se visualiza una lista de preferencias, expanda **Entorno de trabajo** y pulse **Colores y fonts**. Se visualiza el panel Colores y Fonts.
3. Expanda **EGL** y **Editor** y pulse **Font de texto del editor de EGL**.
4. Para elegir entre una lista de fonts y colores, pulse el botón **Cambiar** y haga lo siguiente:

- a. Para cambiar las preferencias de font, seleccione un font, un estilo de font y un tamaño de las listas desplazables.
- b. Para cambiar la preferencia de color, seleccione un color de la lista desplegable.
- c. Marque el recuadro de selección **Tachado** si desea trazar una línea por la mitad del texto.
- d. Marque el recuadro de selección **Subrayado** si desea trazar una línea por debajo del texto.
- e. Puede obtener una vista previa de las selecciones en el recuadro Ejemplo. Cuando haya terminado de realizar las selecciones, pulse **Aceptar**.
5. Para utilizar el font por omisión del sistema operativo, pulse el botón **Utilizar font del sistema**.
6. Para utilizar el font del entorno de trabajo por omisión, pulse el botón **Restablecer**.
7. Para establecer el font para todos los editores (no solo para el editor EGL) en el font por omisión del entorno de trabajo, pulse el botón **Restaurar valores por omisión**.
8. Para guardar los cambios, pulse en **Aplicar** o (si ha terminado de establecer preferencias) pulse en **Aceptar**.

Tareas relacionadas

“Establecer preferencias de EGL” en la página 113

Establecer preferencias para el depurador de EGL

Para establecer preferencias para el depurador de EGL, siga estos pasos:

1. Pulse en **Ventana > Preferencias**.
2. Cuando se visualice una lista, expanda **EGL** y pulse en **Depurar**.
3. Borre o seleccione el recuadro de selección denominado **Solicitar ID de usuario y contraseña de SQL cuando sea necesario**.
Para conocer detalles sobre su elección, consulte *Depurador de EGL*.
4. Borre o seleccione el recuadro de selección denominado **Establecer systemType en DEBUG**.
Para conocer detalles sobre su elección, consulte *Depurador de EGL*.
5. Establezca los valores iniciales para `sysVar.terminalID`, `sysVar.sessionID` y `sysVar.userID`. Si no especifica valores, cada uno de ellos toma por omisión el ID de usuario en Windows 2000/NT/XP o Linux.
6. Establezca el valor de Puerto de depurador EGL. El valor por omisión es 8345.
7. Seleccione el tipo de codificación de caracteres que debe utilizarse al procesar datos durante una sesión de depuración. El valor por omisión es la codificación de archivos del sistema local. Para conocer detalles sobre su elección, consulte el apartado *Opciones de codificación de caracteres para el depurador EGL*.
8. Para especificar clases de Java externas para utilizarlas cuando se ejecuta el depurador, modifique la vía de acceso de clases. Es posible que necesite clases adicionales para dar soporte, por ejemplo, a MQSeries, controladores JDBC o funciones de acceso Java.

Las adiciones a la vía de acceso de clases no son visibles al entorno de prueba de WebSphere Application Server, pero puede añadir a la vía de acceso de clases de ese entorno trabajando en la pestaña Entorno de la configuración de servidor.

Utilice los botones a la derecha del recuadro Orden de vía de acceso de clases:

- Para añadir un proyecto, archivo JAR, directorio o variable, pulse el botón adecuado: **Añadir proyecto**, **Añadir JAR**, **Añadir directorio** o **Añadir variable**.
 - Para eliminar una entrada, selecciónela y pulse en **Eliminar**.
 - Para mover una entrada de una lista de dos o más entradas, seleccione la entrada y pulse en **Colocar antes** o **Colocar después**.
9. Para restaurar los valores por omisión, pulse en **Restaurar valores por omisión**.
 10. Para guardar los cambios, pulse en **Aplicar** o (si ha terminado de establecer preferencias) pulse en **Aceptar**.

Conceptos relacionados

“Compatibilidad con VisualAge Generator” en la página 439

“Depurador EGL” en la página 279

“Opciones de codificación de caracteres para el depurador EGL”

Tareas relacionadas

“Establecer preferencias para conexiones a bases de datos SQL” en la página 119

Consulta relacionada

“sessionID” en la página 934

“terminalID” en la página 938

“userID” en la página 940

Opciones de codificación de caracteres para el depurador EGL

El depurador EGL permite especificar el tipo de codificación de caracteres a utilizar durante la depuración. La codificación de caracteres controla la forma en que el depurador representa los datos de carácter y numéricos internamente, cómo compara los datos de carácter y cómo pasa parámetros a programas, archivos y bases de datos remotos. Para cambiar estas opciones, consulte el apartado *Establecer las preferencias del depurador EGL*.

El depurador EGL da soporte a dos tipos diferentes de codificación de caracteres: la codificación por omisión del sistema local y EBCDIC. La codificación de caracteres por omisión para el depurador EGL es la misma que la del sistema local.

- Si se selecciona la codificación de caracteres por omisión, el depurador representa las variables CHAR, DBCHAR, MBCHAR, DATE, TIME, INTERVAL, NUM y NUMC en el formato por omisión, generalmente ASCII. Las comparaciones entre las variables de carácter utilizan el orden de clasificación ASCII. Los datos deben convertirse a formato de lenguaje principal al llamar a programas remotos y al acceder a archivos y bases de datos remotas.

Si elige este valor y no especifica una tabla de conversión, el depurador elige una tabla de conversión adecuada cuando el usuario llama a un programa remoto o accede a un archivo o base de datos remota. Para obtener más información acerca de las tablas de conversión, consulte el apartado *Conversión de datos*.

- Si se utiliza la codificación de caracteres EBCDIC, el depurador representa las variables CHAR, DBCHAR, MBCHAR, DATE, TIME e INTERVAL con la codificación EBCDIC. Las variables NUM y NUMC se representan en formato numérico de lenguaje principal. Las comparaciones entre las variables de carácter utilizan el orden de clasificación EBCDIC. Los datos no necesitan convertirse a formato de lenguaje principal al llamar a programas remotos o al acceder a archivos y bases de datos remotas, sino que se convierten al formato

Java o ASCII adecuado al efectuar llamadas SQL o llamadas a rutinas C++ locales. La codificación EBCDIC está disponible en varios lenguajes.

Si elige la codificación de caracteres EBCDIC y no especifica una tabla de conversión, el depurador no utiliza ninguna tabla de conversión cuando el usuario llama a un programa remoto o accede a un archivo o base de datos remota. El nombre de programa, de biblioteca y los parámetros pasados se codifican según la codificación de caracteres EBCDIC.

Si la codificación de caracteres seleccionada no está soportada en su JRE (Java Runtime Environment), observará un mensaje de aviso cuando se inicie el depurador. Si opta por continuar la depuración, el depurador volverá al tipo de codificación por omisión.

No puede cambiar la codificación de caracteres durante una sesión de depuración. Debe reiniciar el depurador para que el cambio de la codificación de caracteres entre en vigor.

Conceptos relacionados

“Depurador EGL” en la página 279

Tareas relacionadas

“Establecer preferencias para el depurador de EGL” en la página 114

Consulta relacionada

“Conversión de datos” en la página 467

Establecer los descriptores de construcción por omisión

Para obtener una visión general del descriptor de construcción por omisión y las normas de prioridad del descriptor de construcción, consulte *Generación en el entorno de trabajo*.

Para especificar una preferencia para descriptores de construcción en el nivel del Entorno de trabajo, haga lo siguiente:

1. Pulse en **Ventana > Preferencias**.
2. Cuando se visualice una lista, expanda **EGL** y pulse en **Descriptor de construcción por omisión**.
3. Seleccione **Descriptor de construcción de depuración** y **Descriptor de construcción de sistema destino**.
4. Pulse en **Aplicar** y, a continuación, pulse en **Aceptar**.

Para especificar una preferencia para descriptores de construcción en el nivel de archivo, carpeta, paquete o proyecto, haga lo siguiente:

1. Pulse con el botón derecho del ratón en el nivel de interés (por ejemplo, en el nombre de archivo o carpeta) y, en el menú de contexto, pulse en **Propiedades**.
2. Seleccione **Descriptores de construcción por omisión de EGL**.
3. Seleccione **Descriptor de construcción de depuración** y **Descriptor de construcción de sistema destino**.
4. Pulse en **Aceptar**.

Conceptos relacionados

“Generación en el entorno de trabajo” en la página 330

Establecer preferencias para el editor de EGL

Para especificar las preferencias del editor de EGL, haga lo siguiente:

1. Pulse en **Ventana > Preferencias**.
2. Cuando se visualice una lista, expanda **EGL** y pulse en **Editor**.
3. Para visualizar números de línea al revisar un archivo EGL, seleccione el recuadro de selección **Mostrar números de línea**. Para borrar los números de línea, deseccione el recuadro de selección. El archivo en sí no resulta afectado.
4. Para mostrar subrayados en rojo donde se encuentren errores en el código fuente, seleccione el recuadro de selección **Anotar errores en el texto**. Para borrar esos subrayados, deseccione el recuadro de selección. El archivo en sí no resulta afectado.
5. Para mostrar un indicador de error en rojo en el margen derecho del editor (regla de visión general) siempre que se encuentre un error en el código fuente, seleccione el recuadro de selección **Anotar errores en regla de visión general**. Pulsar en el indicador de error le llevará a la ubicación del error en el código fuente. Para borrar el indicador de error, deseccione el recuadro de selección. El archivo en sí no resulta afectado.
6. Para especificar estilos de fuente, siga el proceso descrito en *Establecer preferencias para estilos de fuente*.
7. Para añadir, eliminar y personalizar plantillas para su uso en la ayuda de contenido, siga el proceso descrito en *Establecer preferencias para plantillas*.
8. Para cambiar la visualización del texto, siga el proceso descrito en la sección *Establecer preferencias para el texto*.

Tareas relacionadas

“Establecer preferencias para estilos de fuente”

“Establecer preferencias para plantillas” en la página 118

“Establecer las preferencias del texto” en la página 113

Consulta relacionada

“Ayuda de contenido en EGL” en la página 483

Establecer preferencias para estilos de fuente

Puede cambiar la manera en que se visualiza el código EGL en el editor de EGL:

1. Pulse en **Ventana > Preferencias**
2. Cuando se visualice una lista de preferencias, expanda **EGL** y **Editor** y, a continuación, pulse en **Estilos de fuente**.
3. Para seleccionar el color que desee que aparezca detrás del tipo de fuente, pulse el botón de selección **Personalizado** en el recuadro de Color de fondo. Pulse en el botón junto a la etiqueta Personalizado. Aparecerá una paleta de color. Seleccione un color y, a continuación, pulse en **Aceptar**.
4. En el recuadro Primer plano, seleccione un tipo de texto y, a continuación, pulse en el botón **Color**. Aparecerá una paleta de color. Seleccione un color y, a continuación, pulse en **Aceptar**.
5. Seleccione el recuadro de selección **Negrita** si desea que el tipo sea negrita.
6. Para guardar los cambios, pulse en **Aplicar** o (si ha terminado de establecer preferencias) pulse en **Aceptar**.

Tareas relacionadas

“Establecer preferencias de EGL” en la página 113

Establecer preferencias para plantillas

Haga lo siguiente para añadir, eliminar o personalizar las plantillas que se visualizan al solicitar la ayuda de contenido en el editor de EGL:

1. Pulse en **Ventana > Preferencias**.
2. Cuando se visualice una lista de preferencias, expanda **EGL** y **Editor** y, a continuación, pulse en **Plantillas**. Se visualiza una lista de plantillas.

Nota: Al igual que en otras aplicaciones de Windows 2000/NT/XP, puede pulsar en una entrada para seleccionarla; puede utilizar **Control-pulsación** para seleccionar o deseleccionar una entrada sin afectar a otras selecciones; y puede utilizar **Despl-pulsación** para seleccionar un conjunto de entradas contiguas a la última entrada en la que ha pulsado.

3. Para hacer que una plantilla esté disponible en el editor de EGL, seleccione el recuadro de selección a la izquierda de un nombre de plantilla. Para hacer que estén disponibles todas las plantillas listadas, pulse en **Habilitar todo**. De forma similar, para hacer que una plantilla no esté disponible, deseccione el recuadro de selección relacionado; y para hacer que todas las plantillas listadas no estén disponibles, pulse en **Inhabilitar todo**.
4. Para crear una nueva plantilla, haga lo siguiente:
 - a. Pulse en **Nueva**
 - b. Cuando aparezca el diálogo Plantilla nueva, especifique un nombre y una descripción, ya que una plantilla se visualizará en una lista de ayuda de contenido solamente si la combinación de nombre y descripción es exclusiva entre todas las plantillas.

Nota: Si la primera palabra utilizada en la plantilla es una palabra clave de EGL (por ejemplo Función), la plantilla estará disponible cuando solicite ayuda de contenido en el editor de EGL, pero solamente cuando el cursor de la pantalla esté en una posición en la que la palabra sea válida. De forma similar, si teclea un prefijo y, a continuación, solicita ayuda de contenido, todas las plantillas que empiecen por ese prefijo estarán disponibles siempre que el cursor en la pantalla esté en posición en que esa plantilla esté permitida sintácticamente. Por ejemplo, teclee "fun" para solicitar plantillas de función. Si no teclea un prefijo o la primera palabra completa, no verá ninguna plantilla al solicitar ayuda de contenido.

- c. En el campo **Patrón**, teclee la propia plantilla:
 - Teclee el texto que desee visualizar
 - Para colocar una variable ya existente en la posición del cursor en la pantalla, pulse en **Insertar variable** y, a continuación, pulse dos veces en una variable. Al insertar la plantilla en el editor de EGL, cada una de esas variables se resuelve al valor adecuado.
 - Para crear una variable personalizada, teclee un signo de dólar (\$) seguido de un corchete izquierdo ({}), una serie de caracteres y un corchete derecho ({}), como en este ejemplo:

```
{variable}
```

Le resultará más fácil insertar una variable ya existente y cambiar el nombre para sus fines.

Al insertar una plantilla personalizada en el editor de EGL, cada variable queda subrayada para indicar que es necesario un valor.

- Para completar la tarea, pulse en **Aceptar** y, en la pantalla de las plantillas, pulse en **Aplicar**.
5. Para revisar una plantilla ya existente, pulse en la entrada listada y revise el recuadro Vista previa.
 6. Para editar una plantilla ya existente, pulse en la entrada listada y, a continuación, pulse en **Editar**. Interactúe con el diálogo Editar plantilla de la misma manera que con el diálogo Plantilla nueva.
 7. Para eliminar una plantilla existente, pulse en la entrada listada y, a continuación, pulse en **Eliminar**. Para eliminar múltiples plantillas, utilice el convenio de Windows 2000/NT/XP para seleccionar múltiples entradas de lista y, a continuación, pulse en **Eliminar**.
 8. Para importar una plantilla desde un archivo XML, pulse en **Importar** a la derecha de la lista de plantillas y siga el mecanismo de examen para especificar la ubicación del archivo.
 9. Para exportar una plantilla a un archivo XML, pulse en **Exportar** a la derecha de la lista de plantillas y siga el mecanismo de examen para especificar la ubicación del nuevo archivo. Para exportar múltiples plantillas, utilice el mecanismo de Windows 2000/NT/XP para seleccionar múltiples entradas de lista y, a continuación, pulse en **Exportar**.
 10. Para exportar todas las plantillas listadas a un archivo XML, pulse en **Exportar todo** y siga el mecanismo de examen para especificar la ubicación del archivo.
 11. Para guardar los cambios, pulse en **Aplicar**. Para volver a la lista de plantillas que estaba en vigor en el momento de la instalación, pulse en **Restaurar valores por omisión**.

Tareas relacionadas

“Utilizar las plantillas EGL con la ayuda de contenido” en la página 129

Establecer preferencias para conexiones a bases de datos SQL

Se utiliza la página para conexiones a bases de datos SQL por estos motivos:

- Puede habilitar el acceso de tiempo de declaración y tiempo de depuración a una base de datos a la que se accede fuera de J2EE.
- Además, puede establecer un valor para la opción del descriptor de construcción sqlJNDIName, que especifica un nombre al que está enlazado el origen de datos por omisión en el registro de JNDI; por ejemplo, java:comp/env/jdbc/MyDB. Esa opción se incluye en el descriptor de construcción que se crea automáticamente en la siguiente situación:
 - Al utilizar el Asistente para Proyecto Web EGL, tal como se describe en *Crear un proyecto para trabajar con EGL*; y
 - Al trabajar en ese asistente, si solicita que se cree un descriptor de construcción.

Haga lo siguiente:

1. Pulse en **Ventana > Preferencias**
2. Cuando se visualice una lista de preferencias, expanda **EGL** y, a continuación, pulse en **Conexiones a base de datos SQL**.
3. En el campo **URL de conexión**, teclee la URL utilizada para conectarse a la base de datos mediante JDBC:

- Para IBM DB2 APP DRIVER para Windows, la URL es `jdbc:db2:dbName` (donde *dbName* es el nombre de base de datos)
- Para el controlador del lado del cliente ligero JDBC Oracle, la URL varía según la ubicación de la base de datos. Si la base de datos es local para la máquina, la URL es `jdbc:oracle:thin:dbName` (donde *dbName* es el nombre de la base de datos). Si la base de datos está en un servidor remoto, la URL es `jdbc:oracle:thin:@host:port:dbName` (donde *host* es el nombre de sistema principal del servidor de base de datos, *port* es el número de puerto y *dbName* es el nombre de la base de datos)
- Para el controlador NET JDBC Informix, la URL es la siguiente (con las líneas combinadas en una):

```
jdbc:informix-sqli://host:port
/dbName:informixserver=servername;
user=userName;password=password
```

host

Nombre de la máquina en que reside el servidor de base de datos

port

Número de puerto

dbName

Nombre de base de datos

serverName

Nombre del servidor de base de datos

userName

ID de usuario de Informix

passWord

La contraseña asociada con el ID de usuario

4. En el campo **Base de datos**, teclee el nombre de la base de datos.
5. En el campo **ID de usuario**, teclee el ID de usuario para la conexión.
6. En el campo **Contraseña**, teclee la contraseña para el ID de usuario.
7. En el campo **Tipo de proveedor de base de datos**, seleccione el producto y versión de base de datos que está utilizando para la conexión JDBC.
8. En el campo **Controlador JDBC**, seleccione el controlador JDBC que está utilizando para la conexión JDBC.
9. En el campo **Clase de controlador JDBC**, teclee la clase de controlador para el controlador que ha seleccionado. Para IBM DB2 APP DRIVER para Windows, la clase de controlador es `COM.ibm.db2.jdbc.app.DB2Driver`; para el controlador del lado del cliente ligero JDBC Oracle, la clase de controlador es `oracle.jdbc.driver.OracleDriver`; y para el controlador NET JDBC Informix, la clase de controlador es `com.informix.jdbc.IfxDriver`. Para otras clases de controlador, consulte la documentación del controlador.
10. En el campo **ubicación de clase**, teclee el nombre de archivo totalmente calificado del archivo *.jar o *.zip que contiene la clase de controlador. Para IBM DB2 APP DRIVER para Windows, teclee el nombre de archivo totalmente calificado para el archivo db2java.zip; por ejemplo, `d:\sql11ib\java\db2java.zip`. Para Oracle THIN JDBC DRIVER, teclee el nombre de archivo totalmente calificado para el archivo classes12.zip; por ejemplo, `d:\0ra81\jdbc\lib\classes12.zip`. Para otras clases de controlador, consulte la documentación del controlador.
11. En el campo **Nombre JNDI de conexión**, especifique la base de datos utilizada en J2EE. El valor es el nombre al que está enlazado el origen de datos en el registro JNDI; por ejemplo, `java:comp/env/jdbc/MyDB`. Como se

ha indicado anteriormente, este valor se asigna a la opción `sqlJNDIName` en el descriptor de construcción que se construye automáticamente para un proyecto Web EGL dado.

12. Si está accediendo a DB2 UDB y especifica un valor en el campo **ID de autenticación secundario**, el valor se utiliza en la sentencia SET CURRENT SQLID utilizada por EGL en el momento de la validación. El valor es sensible a las mayúsculas y minúsculas.

Puede borrar o aplicar valores de preferencia:

- Para restaurar valores por omisión, pulse en **Restaurar valores por omisión**.
- Para aplicar valores de preferencia sin salir del diálogo de preferencias, pulse en **Aplicar**.
- Si ha terminado de establecer preferencias, pulse **Aceptar**.

Tareas relacionadas

“Crear un proyecto Web EGL” en la página 126

“Establecer preferencias de EGL” en la página 113

Consulta relacionada

“sqlJNDIName” en la página 402

Establecer preferencias para la recuperación de SQL

En el momento de la declaración de EGL, puede utilizar la característica de recuperación de SQL para crear un registro de SQL a partir de las columnas de una tabla de SQL. Para obtener una visión general, consulte *Soporte de SQL*.

Para establecer preferencias para la característica de recuperación de SQL, haga lo siguiente:

1. Pulse en **Ventana > Preferencias** y, a continuación, expanda **EGL** y pulse en **Recuperación de SQL**
2. Especifique reglas para crear cada elemento de estructura que cree la característica de recuperación de SQL:
 - a. Para especificar el tipo EGL que se debe utilizar al crear un elemento de estructura de un tipo de datos de caracteres SQL, pulse uno de los botones de selección siguientes:
 - **Utilizar tipo string de EGL** (el valor predeterminado) correlaciona tipos de datos char de SQL con tipos de datos string de EGL
 - **Utilizar tipo char de EGL** correlaciona los tipos de datos char de SQL con los tipos de datos char de EGL
 - **Utilizar tipo mbChar de EGL** correlaciona tipos de datos char de SQL con tipos de datos mbChar de EGL
 - **Utilizar tipo Unicode de EGL** correlaciona tipos de datos char de SQL con tipos de datos Unicode de EGL
 - b. Para especificar mayúsculas/minúsculas en el nombre del elemento de estructura, pulse en uno de los siguientes botones de selección:
 - **No cambiar mayúsculas/minúsculas** (el valor por omisión) significa que las mayúsculas/minúsculas del elemento de estructura son las mismas que las del nombre de columna de tabla relacionado
 - **Cambiar a minúsculas** significa que el nombre del elemento de estructura es una versión en minúsculas del nombre de columna de tabla
 - **Cambiar a minúsculas y escribir en mayúsculas primera letra tras subrayado** también significa que el nombre del elemento de estructura es

una versión en minúsculas del nombre de columna de tabla, excepto en que una letra del nombre del elemento de estructura se representa en mayúsculas si, en el nombre de columna de tabla, la letra va justo después de un subrayado

c. Para especificar cómo los subrayados del nombre de columna de tabla quedan reflejados en el nombre de elemento de estructura, pulse en uno de los siguientes botones de selección:

- **No cambiar subrayados** (el valor por omisión) significa que los subrayados del nombre de columna de tabla se incluyen en el nombre de elemento de estructura
- **Eliminar subrayados** significa que los subrayados del nombre de columna de tabla no se incluyen en el nombre de elemento de estructura
- **Cambiar subrayados por guiones** significa que los subrayados del nombre de columna de tabla se representan como guiones en el nombre de elemento de estructura

3. Si tiene intención de recuperar datos de una tabla que forma parte de un esquema del sistema Informix, deseccione el recuadro de selección para **Excluir esquemas de sistema**. (En este caso, "Informix" es el propietario de la tabla.) En todos los demás casos, seleccione el recuadro de selección para mejorar el rendimiento de la característica de recuperación de SQL.

El recuadro de selección se selecciona por omisión.

Conceptos relacionados

"Soporte de SQL" en la página 229

Tareas relacionadas

"Recuperar datos de tabla SQL" en la página 252

"Establecer preferencias de EGL" en la página 113

"Establecer preferencias para conexiones a bases de datos SQL" en la página 119

Consulta relacionada

"Informix y EGL" en la página 252

Habilitar posibilidades de EGL

Para acceder a la funcionalidad de EGL, deben habilitarse las posibilidades de EGL. Están disponibles las posibilidades de EGL siguientes:

Desarrollo de EGL

Consta de todas las funciones relacionadas con el desarrollo y la depuración de aplicaciones de EGL.

Migración de EGL V6.0

Consta de toda la funcionalidad relacionada con la conversión para ajustarse a EGL V6.0 iFix del código fuente de EGL 5.1.2 y 6.0.

Migración de VisualAge Generator a EGL

Consta de todas las funciones relacionadas con la migración de código existente de VisualAge Generator a código EGL.

Para habilitar las posibilidades de EGL, haga lo siguiente:

1. Pulse en **Ventana > Preferencias**.
2. Cuando se visualice una lista de preferencias, expanda **Entorno de trabajo** y pulse **Posibilidades**. Se visualiza el panel Posibilidades.

3. Si desea recibir una solicitud la primera vez que se utiliza una característica que necesita habilitar una posibilidad, marque el recuadro de selección **Solicitar al habilitar posibilidades**.
4. Expanda la carpeta de posibilidades **Desarrollador EGL**.
5. Marque el recuadro de selección para las posibilidades de EGL deseadas. Como alternativa, puede seleccionar la carpeta de posibilidades **Desarrollador EGL** para habilitar todas las posibilidades que contiene la carpeta.
6. Para devolver la lista de posibilidades habilitadas a el estado correspondiente a la instalación del producto, pulse el botón **Restaurar valores por omisión**.
7. Para guardar los cambios, pulse **Aplicar** y después **Aceptar**.

Nota: Al habilitar las posibilidades de EGL, se habilitarán automáticamente otras posibilidades necesarias para desarrollar y depurar aplicaciones de EGL.

Tareas relacionadas

“Establecer preferencias de EGL” en la página 113

Iniciar el desarrollo de código

Crear un proyecto

Crear un proyecto EGL

Para obtener una visión general de cómo organizar el trabajo, consulte *Proyectos, paquetes y archivos de EGL*.

Para preparar un proyecto EGL nuevo, haga lo siguiente:

1. En el Entorno de trabajo, realice uno de estos dos pasos:
 - Pulse en **Archivo > Nuevo > Proyecto**; o bien
 - Pulse con el botón derecho del ratón y, a continuación, pulse en **Nuevo > Proyecto**.

Se abre el asistente Proyecto nuevo.

2. Expanda **EGL** y pulse **Proyecto EGL**. Pulse en **Siguiente**. Se visualizará el asistente de **Proyecto EGL nuevo**.

Nota: Si el proyecto EGL no está disponible, marque el recuadro de selección **Mostrar todos los asistentes**.

3. En el campo **Nombre de proyecto**, teclee un nombre para el proyecto. Por omisión, el proyecto se coloca en su espacio de trabajo, pero puede pulsar en **Examinar** y elegir una ubicación distinta.
4. Seleccione cómo especificar un descriptor de construcción, que es el componente que dirige el proceso en el momento de la generación:
 - **Crear descriptor(es) de construcción de proyecto nuevo automáticamente** significa que EGL proporciona descriptores de construcción y los graba en un archivo de construcción (extensión .eglbld) que tiene el mismo nombre que el proyecto.

Para especificar algunos de los valores en esos descriptores de construcción, pulse en **Opciones**. Para cambiar esos valores posteriormente, cambie el archivo de construcción que se ha creado automáticamente.

Para obtener más detalles, consulte *Especificar opciones de base de datos en la creación del proyecto*.

- **Utilizar un descriptor de construcción especificado en preferencia de EGL** significa que EGL señala a un descriptor de construcción que ha creado e identificado como una preferencia de EGL.
 - **Seleccionar descriptor de construcción existente** le permite especificar un descriptor de construcción de los disponibles en el espacio de trabajo.
5. En la mayoría de casos, pulse en **Finalizar**. No obstante, si pulsa en **Siguiente**, puede especificar otras carpetas y proyectos origen para referenciar desde el proyecto que está creando. Cuando haya terminado de seleccionar otras carpetas y proyectos origen, pulse en **Finalizar**.

Conceptos relacionados

“Componente descriptor de construcción” en la página 293

“Proyectos, paquetes y archivos EGL” en la página 13

Tareas relacionadas

“Especificar opciones de base de datos durante la creación del proyecto” en la página 127
“Establecer preferencias para conexiones a bases de datos SQL” en la página 119

Crear un proyecto Web EGL

Para obtener una visión general de cómo organizar el trabajo, consulte *Proyectos, paquetes y archivos de EGL*.

Para preparar un proyecto Web de EGL Web nuevo, haga lo siguiente:

1. En el Entorno de trabajo, realice uno de estos dos pasos:
 - Pulse en **Archivo > Nuevo > Proyecto**; o bien
 - Pulse con el botón derecho del ratón y, a continuación, pulse en **Nuevo > Proyecto**.

Se abre el asistente Proyecto nuevo.
2. Expanda **EGL** y pulse **Proyecto Web EGL**. Pulse en **Siguiente**. Se visualizará el asistente de **Proyecto Web EGL nuevo**.
3. En el campo **Nombre de proyecto**, teclee un nombre para el proyecto. Por omisión, el proyecto se coloca en su espacio de trabajo; pero puede pulsar en **Examinar** y elegir una ubicación distinta.
4. Seleccione cómo especificar un descriptor de construcción, que es el componente que dirige el proceso en el momento de la generación:
 - **Crear descriptor(es) de construcción de proyecto nuevo automáticamente** significa que EGL proporciona descriptores de construcción y los graba en un archivo de construcción (extensión .eglbld) que tiene el mismo nombre que el proyecto.

Para especificar algunos de los valores en esos descriptores de construcción, pulse en **Opciones**. Para cambiar esos valores posteriormente, cambie el archivo de construcción que se ha creado automáticamente.

Para obtener más detalles, consulte *Especificar opciones de base de datos en la creación del proyecto*.
 - **Utilizar un descriptor de construcción especificado en preferencia de EGL** significa que EGL señala a un descriptor de construcción que ha creado e identificado como una preferencia de EGL.
 - **Seleccionar descriptor de construcción existente** le permite especificar un descriptor de construcción de los disponibles en el espacio de trabajo.
5. Si ha solicitado que se cree un descriptor de construcción automáticamente, puede colocar un valor en el campo **Nombre JNDI para conexión SQL**. El efecto es asignar el nombre al que está enlazado el origen de datos por omisión en el registro de JNDI en el momento de la depuración o la generación. (Un valor de ejemplo es `java:comp/env/jdbc/MyDB`.) La selección asigna un valor a la opción del descriptor de construcción `sqlJNDIName`. Si el campo **Nombre JNDI para conexión SQL** ya está relleno, el valor se habrá obtenido de una preferencia del Entorno de trabajo, tal como se describe en *Establecer preferencias para conexiones a base de datos SQL*.
6. En la mayoría de casos, pulse en **Finalizar**. Para personalizar (lo que es posible para cualquier proyecto Web), configure los valores de J2EE en la parte inferior del diálogo. Opcionalmente, puede pulsar **Ocultar avanzados** para ocultar los valores J2EE. Pulse en **Siguiente**. Seleccione valores de característica y, a continuación, pulse en **Siguiente**. Seleccione una plantilla de página y, a continuación, pulse en **Finalizar**.

Conceptos relacionados

“Componente descriptor de construcción” en la página 293

“Proyectos, paquetes y archivos EGL” en la página 13

Tareas relacionadas

“Especificar opciones de base de datos durante la creación del proyecto”

“Establecer preferencias para conexiones a bases de datos SQL” en la página 119

Consulta relacionada

“sqlJNDIName” en la página 402

Especificar opciones de base de datos durante la creación del proyecto

Para asignar valores de opciones en el descriptor de construcción que EGL crea automáticamente, trabaje en el diálogo **Opciones de creación de proyectos**. Para conocer detalles sobre cómo visualizar el diálogo, consulte *Crear un proyecto para trabajar con EGL*.

Para aceptar la información de conexión a base de datos que se ha especificado en preferencias, pulse en el recuadro de selección.

La siguiente tabla muestra cada etiqueta en pantalla y la opción del descriptor de construcción relacionada.

Etiqueta	Opción del descriptor de construcción
Tipo de base de datos	dbms
Controlador JDBC de base de datos	sqlJDBCClass
Nombre de base de datos	sqlJNDIName (para salida de J2EE) o sqlDB (para salida no de J2EE)

Conceptos relacionados

“Componente descriptor de construcción” en la página 293

Tareas relacionadas

“Crear un proyecto Web EGL” en la página 126

Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

Crear una carpeta fuente de EGL

Una vez cree un proyecto en el entorno de trabajo, puede crear una o varias carpetas dentro de ese proyecto para que contengan sus archivos de EGL.

Para crear una carpeta para agrupar archivos de EGL, haga lo siguiente:

1. En el entorno de trabajo pulse en **Archivo > Nuevo > Carpeta fuente EGL**.
2. Seleccione el proyecto que contendrá la carpeta EGL. En el campo Nombre de carpeta, teclee el nombre de la carpeta de EGL, por ejemplo myFolder.
3. Pulse en el botón **Finalizar**.

Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13

“Introducción a EGL” en la página 1

Tareas relacionadas

“Crear un proyecto Web EGL” en la página 126

Consulta relacionada

“Crear un archivo fuente EGL”

“Convenios de denominación” en la página 672

Crear un paquete de EGL

Un paquete de EGL es una colección de componentes fuente relacionados con nombre. Para crear un paquete de EGL, haga lo siguiente:

1. Identifique un proyecto o carpeta para que contenga el paquete. Debe crear un proyecto o carpeta si no tiene uno todavía.
2. En el entorno de trabajo pulse en **Archivo > Nuevo > Paquete EGL**.
3. Seleccione el proyecto o carpeta que contendrá el paquete de EGL. La carpeta origen ya podría tener contenido dependiendo de la selección actual en el Explorador de proyectos.
4. En el campo Nombre de paquete, teclee el nombre del paquete de EGL. Consulte *Proyectos, paquetes y archivos de EGL* para conocer detalles sobre los convenios de denominación de paquetes.
5. Pulse en el botón **Finalizar**.

Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13

“Introducción a EGL” en la página 1

Tareas relacionadas

“Crear una carpeta fuente de EGL” en la página 127

“Crear un proyecto Web EGL” en la página 126

Consulta relacionada

“Crear un archivo fuente EGL”

Crear un archivo fuente EGL

Para crear un archivo fuente EGL, haga lo siguiente:

1. Identifique un proyecto o carpeta para que contenga el archivo. Debe crear un proyecto o carpeta si no tiene uno todavía.
2. En el entorno de trabajo, pulse en **Archivo > Nuevo > Archivo fuente EGL**.
3. Seleccione el proyecto o carpeta que contendrá el archivo EGL. Seleccione el paquete que contendrá el archivo EGL. En el campo Nombre de archivo fuente EGL, teclee el nombre del archivo EGL, por ejemplo myEGLFile.
4. Pulse en **Finalizar** para crear el archivo. Se añade una extensión (.egl) automáticamente al final del nombre de archivo. El archivo EGL aparece en la vista Explorador de proyectos y se abre automáticamente en el editor de EGL por omisión.

Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13

“Introducción a EGL” en la página 1

Tareas relacionadas

“Crear una carpeta fuente de EGL” en la página 127

“Crear un proyecto Web EGL” en la página 126

Utilizar las plantillas EGL con la ayuda de contenido

Para practicar con la ayuda de contenido, haga lo siguiente:

1. Abra un archivo de EGL nuevo.
2. En una línea disponible, teclee **P** (para PageHandler o programa) y pulse **Control + espacio**.
3. Cuando aparezca una ventana emergente, pulse en un icono del componente a personalizar. Realice uno de estos pasos:
 - Pulse **Intro** para seleccionar el primer icono de la lista; o bien
 - Utilice las teclas de flechas para seleccionar otro icono (para un programa) y pulse en **Intro**.

El editor coloca una plantilla de componente en el archivo.

4. Personalice el componente.

Cuando se visualice la plantilla, el editor resalta la primera área en la que debe teclear información; en este caso, especifique el nombre de componente. Tras teclearlo, pulse el **Tabulador** para resaltar la siguiente área en la que debe teclear.

Puede utilizar la tecla **Tabulador** repetidamente y este uso de la tecla está disponible hasta llegar al final del archivo o hasta que cambie su posición en el archivo de cualquier otra manera.

5. Para insertar una función en el programa o PageHandler, teclee **F** (para función) y, a continuación, pulse **Control + espacio**. Aunque puede volver a seleccionar una plantilla de componente, haga lo siguiente:
 - Utilice las teclas de flechas o el ratón para desplazarse hasta el final de la lista
 - Pulse **Intro** o pulse en la palabra *Función*; observe que la ausencia de un icono significa que está seleccionando una serie en lugar de una plantilla de componente

La capacidad de seleccionar una serie es más útil en otros contextos, por ejemplo cuando desea teclear un nombre de variable rápidamente.

6. Con el cursor al final de la palabra *Función*, pulse **Control + espacio** y pulse en un icono de la lista.

El editor coloca la plantilla de función en el archivo.

7. Personalice el componente.
8. A medida que desarrolle el código, pulse **Control + espacio** periódicamente para comprender el rango de servicios proporcionado.

Tareas relacionadas

“Insertar fragmentos de código en archivos EGL y JSP” en la página 149

“Establecer preferencias para plantillas” en la página 118

Consulta relacionada

“Componente de función en formato fuente EGL” en la página 527

“Componente PageHandler en formato fuente EGL” en la página 679

“Componente de programa en formato fuente EGL” en la página 728

Accesos directos para EGL

La tabla siguiente muestra los accesos directos de teclado disponibles en el editor EGL.

Combinación de clave	Función
Ctrl+/	Comentario
Ctrl+\	Descomentar
Ctrl+A	Seleccionar todo
Ctrl+C	Copiar
Ctrl+F	Buscar
Ctrl+H	Buscar
Ctrl+K	Buscar siguiente
Ctrl+S	Guardar
Ctrl+V	Pegar
Ctrl+X	Cortar
Ctrl+G	Generar
Ctrl+L	Ir a una línea específica
Ctrl+Y	Rehacer
Ctrl+Z	Deshacer
Ctrl+Mayús+A	Añade una sentencia SQL explícita a una sentencia de E/S de EGL que tiene una implícita
Ctrl+Mayús+K	Buscar anterior
Ctrl+Mayús+N	Accede al diálogo Abrir componente
Ctrl+Mayús+P	Construir una sentencia EGL prepare y la sentencia get , execute u open
Ctrl+Mayús+R	Utilizar la característica para crear o sobrescribir elementos en un componente de registro SQL
Ctrl+Mayús+S	Mostrar el archivo actual en el Explorador de proyectos
Ctrl+Mayús+V	Ver y validar la sentencia SQL asociada con una sentencia EGL de E/S y realizar acciones relacionadas
Ctrl+Espacio	Obtener ayuda de contenido
F3	Abrir el archivo que contiene el componente cuyo nombre está resaltado
Tabulador	Sangra el texto hasta el próximo tope de tabulador

Desarrollar código fuente EGL básico

Crear un componente EGL dataItem

Un componente dataItem de EGL define un área de memoria que no puede dividirse. Los componentes dataItem de EGL están contenidos en archivos de EGL. Para crear un componente dataItem de EGL, haga lo siguiente:

1. Busque un archivo EGL para que albergue el componente dataItem y abra el archivo en el editor EGL. Debe crear un archivo EGL si todavía no tiene uno.
2. Teclee las especificaciones del componente dataItem de acuerdo con la sintaxis de EGL (para conocer los detalles, consulte la sección *Componente DataItem en formato de código fuente EGL*). Puede utilizar la ayuda de contenido para incluir un esquema de la sintaxis del componente dataItem en el archivo.
3. Guarde el archivo EGL.

Conceptos relacionados

“Componente dataItem”

“Proyectos, paquetes y archivos EGL” en la página 13

Tareas relacionadas

“Crear un archivo fuente EGL” en la página 128

“Utilizar las plantillas EGL con la ayuda de contenido” en la página 129

Consulta relacionada

“Ayuda de contenido en EGL” en la página 483

“Componente DataItem en formato fuente EGL” en la página 472

“Convenios de denominación” en la página 672

Componente dataItem

Un *componente dataItem* define un área de memoria que no puede subdividirse. Un componente dataItem es un componente autónomo, a diferencia de un campo de estructura dentro de una estructura fija.

Una *variable primitiva* es un área de memoria basado en un componente dataItem o en una declaración primitiva como por ejemplo INT o CHAR(2). Puede utilizar una variable primitiva de las siguientes maneras:

- Como parámetro que recibe datos en una función o programa
- Como variable de una función EGL; por ejemplo, en una sentencia assignment como argumento que pasa datos a otra función o programa

Cada variable primitiva tiene una serie de propiedades, que pueden estar establecidas por omisión o pueden especificarse en la variable o en el componente dataItem. Para obtener información detallada, consulte la sección *Visión general de las propiedades y alteraciones temporales de EGL*.

Conceptos relacionados

“Componentes de registro fijo” en la página 133

“Estructura fija” en la página 26

“Visión general de las propiedades de EGL” en la página 64

“Componentes” en la página 17

“Componentes de registro”

“Typedef” en la página 27

Tareas relacionadas

“Establecer preferencias para plantillas” en la página 118

Consulta relacionada

“Componente DataItem en formato fuente EGL” en la página 472

“Formato fuente EGL” en la página 491

“Inicialización de datos” en la página 471

“Tipos primitivos” en la página 34

Crear un componente de registro EGL

Un componente de registro define una estructura (una disposición jerárquica de elementos de datos de tamaño fijo en almacenamiento) y un enlace opcional, que es una relación, del registro a un origen de datos externo (archivo, base de datos o cola de mensajes). Los componentes de registro de EGL están contenidos en archivos de EGL. Para crear un componente de registro de EGL, haga lo siguiente:

1. Busque un archivo EGL para que albergue el componente de registro y abra el archivo en el editor EGL. Debe crear un archivo EGL si todavía no tiene uno.
2. Teclee las especificaciones del componente de registro de acuerdo con la sintaxis de EGL (para conocer detalles, consulte la sección *Componente de registro básico en formato de código fuente EGL*, *Componente de registro indexado en formato de código fuente EGL*, *Componente de registro MQ en formato de código fuente EGL*, *Componente de registro relativo en formato de código fuente EGL*, *Componente de registro de serie en formato de código fuente EGL* y *Componente de registro SQL en formato de código fuente EGL*). Puede utilizar la ayuda de contenido para incluir un esquema de la sintaxis del componente de registro en el archivo.
3. Guarde el archivo EGL.

Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13

“Componentes de registro”

Tareas relacionadas

“Crear un archivo fuente EGL” en la página 128

“Utilizar las plantillas EGL con la ayuda de contenido” en la página 129

Consulta relacionada

“Componente de registro básico en formato fuente EGL” en la página 379

“Ayuda de contenido en EGL” en la página 483

“Componente de registro indexado en formato fuente EGL” en la página 535

“Componente de registro MQ en formato fuente EGL” en la página 662

“Convenios de denominación” en la página 672

“Componente de registro relativo en formato fuente EGL” en la página 740

“Componente de registro serie en formato fuente EGL” en la página 743

“Componente de registro SQL en formato fuente EGL” en la página 748

Componentes de registro

Un *componente de registro* define una secuencia de datos cuya longitud no es necesariamente conocida durante la generación y que está compuesta de campos. En EGL, un campo define una variable en cualquier registro basado en el componente de registro.

Un campo puede ser un diccionario, un arrayDictionary o una matriz de diccionarios o arrayDictionaries o puede estar basado en cualquiera de los elementos siguientes:

- Un tipo primitivo como por ejemplo STRING
- Un componente DataItem
- Un registro fijo (tal como se describe más adelante)
- Otro componente de registro
- Una matriz de cualquiera de las clases precedentes

Hay dos tipos de componentes de registro disponibles:

- basicRecord, tal como se utiliza para el proceso general pero no para acceder a un almacén de datos
- SQLRecord, tal como se utiliza para acceder a una base de datos relacional

Puede utilizar un *record* en los contextos siguientes:

- En una sentencia que copia datos a o de una base de datos relacional
- En una sentencia assignment o move
- Como argumento que pasa datos a otro programa o función
- Como parámetro que recibe datos en un programa o función

Un componente de registro es distinto de un componente de registro fijo, lo que define una secuencia de datos cuya longitud *se* conoce durante la generación. Un componente de registro fijo se utiliza principalmente para acceder a archivos VSAM, colas de mensajes MQSeries y otros archivos de secuencia.

Un componente de registro que incluye números de nivel es un componente de registro fijo, incluso si el componente de registro es del tipo basicRecord o SQLRecord. Para obtener más detalles, consulte la sección *Componentes de registro fijo*.

Conceptos relacionados

“Componente dataItem” en la página 131

“Componentes de registro fijo”

“Componentes” en la página 17

“Tipos de registros y propiedades” en la página 135

“Asociaciones de recursos y tipos de archivo” en la página 304

“Estructura fija” en la página 26

“Typedef” en la página 27

Tareas relacionadas

“Establecer los descriptores de construcción por omisión” en la página 116

“Establecer preferencias para el editor de EGL” en la página 117

Consulta relacionada

“Formato fuente EGL” en la página 491

“Inicialización de datos” en la página 471

“Tipos primitivos” en la página 34

Componentes de registro fijo

Un componente de registro fijo define una secuencia de datos cuya longitud se conoce durante la generación. Esta clase de componente se compone necesariamente de una serie de campos primitivos, de longitud fija y cada campo

puede subestructurarse. Un campo que especifica un número de teléfono, por ejemplo, puede definirse de la manera siguiente:

```
10 phoneNumber CHAR(10);
20 areaCode CHAR(3);
20 localNumber CHAR(7);
```

Aunque puede utilizar registros fijos (que son variables) para cualquier clase de proceso, se utilizan principalmente para operaciones de E/S en archivos VSAM, colas de mensajes MQSeries y otros archivos secuenciales. Aunque puede utilizar registros fijos para acceder a bases de datos relacionales o para proceso general (tal como era el caso con productos anteriores como por ejemplo VisualAge Generator), debe evitar la utilización de registros fijos para esos propósitos en los desarrollos nuevos.

Un componente de registro de cualquiera de los tipos siguientes es un componente de registro fijo:

- indexedRecord
- mqRecord
- relationalRecord
- serialRecord

Además, un componente de registro de cualquiera de los tipos siguientes es un componente de registro fijo si cada campo va precedido de un número de nivel:

- basicRecord
- SQLRecord

Puede utilizar un registro fijo en el contexto siguiente:

- En una sentencia que copia datos a o de un origen de datos
- En una sentencia assignment o move
- Como argumento que pasa datos a otro programa o función
- Como parámetro que recibe datos en un programa o función

Cualquier relación de un componente de registro fijo con un origen de datos externo viene determinado por el tipo del componente de registro fijo y por un conjunto de propiedades específicas de registro como fileName. Un registro basado en un componente de tipo indexedRecord, por ejemplo, se utiliza para acceder a un Conjunto de datos de secuencia de clave VSAM. La relación de un componente de registro con un origen de datos determina las operaciones generadas cuando el registro fijo se utiliza en una sentencia E/S de EGL como por ejemplo **add**.

Un campo de registro fijo puede estar basado en otro componente de registro fijo y, en sentencias de asignación ese campo se trata como un área de memoria de tipo CHAR independientemente de los tipos en el componente de registro fijo.

Conceptos relacionados

“Componente dataItem” en la página 131

“Componentes de registro” en la página 132

“Tipos de registros y propiedades” en la página 135

“Asociaciones de recursos y tipos de archivo” en la página 304

“Estructura fija” en la página 26

“Typedef” en la página 27

Tareas relacionadas

“Establecer los descriptores de construcción por omisión” en la página 116

“Establecer preferencias para el editor de EGL” en la página 117

Consulta relacionada

“Asignaciones” en la página 374

“Formato fuente EGL” en la página 491

“Inicialización de datos” en la página 471

“Tipos primitivos” en la página 34

Tipos de registros y propiedades

Están disponibles varios tipos de registros EGL:

- basicRecord
- indexedRecord
- mqRecord
- relativeRecord
- serialRecord
- SQLRecord

Para obtener información detallada sobre qué sistemas destino dan soporte a qué tipos de registros, consulte la sección *Referencia cruzada de tipos de archivos y registros*. Para obtener información detallada sobre cómo se inicializan los componentes de registro, consulte la sección *Inicialización de datos*.

basicRecord

Un registro básico o un registro básico fijo se utiliza para el proceso interno y no puede acceder al almacenamiento de datos.

Por omisión, el componente es un componente de registro, pero es un componente de registro fijo las definiciones de campo van precedidas por números de nivel.

En un componente de registro fijo del tipo basicRecord, la propiedad **redefines** está disponible. Si se establece, esa propiedad identifica un registro declarado y cualquier registro basado en el componente de registro fijo accederá a la memoria de tiempo de ejecución del registro declarado.

En un programa principal, la propiedad de programa **inputRecord** identifica un registro (o un registro fijo) que se inicializa automáticamente, como se describe en la sección *Inicialización de datos*.

indexedRecord

Un registro indexado es un registro fijo que le permite trabajar con un archivo al que se accede mediante un *valor de clave*, que identifica la posición lógica de un registro en el archivo. Puede leer el archivo invocando una sentencia **get**, **get next** o **get previous**. Además, puede escribir en el archivo invocando una sentencia **add** o **replace**; y puede eliminar un registro del archivo invocando una sentencia **delete**.

Las propiedades de un componente de tipo indexedRecord son las siguientes:

- La propiedad **fileName** es obligatoria. Para obtener detalles sobre el significado de la entrada, consulte la sección *Asociaciones de recursos (visión general)*. Para obtener detalles sobre los caracteres válidos, consulte la sección *Convenios de denominación*.

- La propiedad **keyItem** es obligatoria pero sólo puede ser un campo de estructura que sea exclusivo en el mismo registro. Debe utilizar una referencia no calificada para especificar el campo de clave; por ejemplo, utilice *myItem* en lugar de *myRecord.myItem*. (Sin embargo, en una sentencia EGL, puede hacer referencia al campo de clave al igual que haría con cualquier campo.)

Consulte también la sección *Propiedades que soportan registros de longitud variable*.

mqRecord

Un registro MQ es un registro fijo que permite acceder a una cola de mensajes MQSeries. Para obtener información detallada, consulte la sección *Soporte de MQSeries*.

relativeRecord

Un registro relativo es un registro fijo que permite trabajar con un conjunto de datos cuyos registros tienen las siguientes propiedades:

- Tienen una longitud fija
- Se puede acceder a ellos mediante un entero que representa la posición secuencial del registro en el archivo

Las propiedades de un componente de tipo *relativeRecord* son las siguientes:

- La propiedad **fileName** es obligatoria. Para obtener detalles sobre el significado de la entrada, consulte la sección *Asociaciones de recursos (visión general)*. Para obtener detalles sobre los caracteres válidos, consulte la sección *Convenios de denominación*.
- La propiedad **keyItem** es obligatoria. El campo de clave puede ser cualquiera de estas áreas de memoria:
 - Un campo de estructura en el mismo registro
 - Un campo de estructura de un registro que es global al programa o que es local a la función que accede al registro
 - Una variable primitiva que es global al programa o que es local a la función que accede al registro

Debe utilizar una referencia no calificada para especificar el campo de clave. Por ejemplo, utilice *myItem* en lugar de *myRecord.myItem*. (En una sentencia EGL, puede hacer referencia al campo de clave de la misma manera que haría referencia a cualquier campo). El campo de clave debe ser exclusivo en el ámbito local de la función que accede al registro o bien debe estar ausente del ámbito local y ser exclusivo en el ámbito global.

El campo de clave tiene las siguientes características:

- Tiene un tipo primitivo de BIN, DECIMAL, INT o NUM
- No contiene posiciones decimales
- Permite 9 dígitos como máximo

Sólo las sentencias **get** y **add** utilizan el campo de clave, pero éste debe estar disponible en cualquier función que utiliza el registro para acceder a archivos.

serialRecord

Un registro serie es un registro fijo que le permite acceder a un archivo o conjunto de datos al que se accede secuencialmente. Puede leer el archivo invocando una sentencia **get**, y una serie de sentencias **get next** lee secuencialmente los registros del archivo, desde el primero al último. Puede escribir en el archivo invocando una sentencia **add**, que coloca un registro nuevo al final del archivo.

Las propiedades del registro serie incluyen **fileName**, que es obligatoria. Para obtener detalles sobre el significado de la entrada para esta propiedad, consulte la sección *Asociaciones de recursos (visión general)*. Para obtener detalles sobre los caracteres válidos, consulte la sección *Convenios de denominación*.

Consulte también la sección *Propiedades que soportan registros de longitud variable*.

sqlRecord

Un registro SQL es un registro (o un registro fijo) que proporciona servicios especiales al acceder a una base de datos relacional.

Por omisión, el componente es un componente de registro, pero es un componente de registro fijo las definiciones de campo van precedidas por números de nivel.

Cada componente tiene las propiedades siguientes:

- Una entrada en **tableNames** identifica una tabla SQL asociada al componente. Puede hacer referencia a varias tablas de una unión, pero las restricciones aseguran que no escribe en varias tablas con una única sentencia EGL. Puede asociar un determinado nombre de tabla con una *etiqueta*, que es un nombre corto opcional que se utiliza para hacer referencia a la tabla en una sentencia SQL.
- **defaultSelectCondition** es opcional. Las condiciones forman parte de la cláusula WHERE de las sentencias SQL por omisión. La cláusula WHERE es significativa cuando se utiliza un registro SQL en una sentencia EGL **open** o **get** o en las sentencias como **get next** o **get previous**.
En la mayoría de casos, la condición de selección por omisión SQL complementa una segunda condición, que se basa en una asociación entre los valores de campo de clave del registro SQL y las columnas de clave de la tabla SQL.
- **tableNameVariables** es opcional. Puede especificar una o más variables cuyo contenido durante la ejecución determina las tablas de base de datos a las que se debe acceder, como se describe en la sección *SQL dinámico*.
- **keyItems** es opcional. Cada campo de clave sólo puede ser un campo de estructura exclusivo en el mismo registro. Debe utilizar una referencia no calificada para especificar cada uno de esos campos; por ejemplo, utilice *myItem* en lugar de *myRecord.myItem*. (Sin embargo, en una sentencia EGL, puede hacer referencia a un campo de clave al igual que haría con cualquier campo.)

Para obtener información detallada, consulte la sección *Soporte de SQL*.

Conceptos relacionados

“Componentes de registro fijo” en la página 133

“SQL dinámico” en la página 240

“Soporte de MQSeries” en la página 265

“Componentes de registro” en la página 132

“Asociaciones de recursos y tipos de archivo” en la página 304

“Soporte de SQL” en la página 229

Consulta relacionada

“add” en la página 561

“close” en la página 568

“Inicialización de datos” en la página 471

“delete” en la página 571

“execute” en la página 574

“get” en la página 585

“get next” en la página 597
“get previous” en la página 602
“Propiedades de registros MQ” en la página 665
“Convenios de denominación” en la página 672
“open” en la página 616
“prepare” en la página 630
“Propiedades que dan soporte a registros de longitud variable” en la página 737
“Referencias cruzadas de tipo de registro y tipo de archivo” en la página 737
“replace” en la página 632
“Propiedades de elementos SQL” en la página 67
“terminalID” en la página 938

Crear un componente de programa de EGL

Un componente de programa de EGL es la unidad lógica principal utilizada para generar un programa Java, una envoltura Java o un bean de sesión Enterprise JavaBean. Para obtener más información, consulte *Componente de programa*.

Se añade un componente de programa automáticamente a un archivo de programa y se denomina adecuadamente al crear el archivo en el entorno de trabajo. Las especificaciones de archivo de programa permiten solamente un componente de programa por archivo y requieren un nombre de programa que coincida con el nombre de archivo.

Para crear un archivo de programa con un componente de programa, haga lo siguiente:

1. Identifique un proyecto o carpeta para que contenga el archivo. Debe crear un proyecto o carpeta si no tiene uno todavía.
2. En el entorno de trabajo, pulse en **Archivo > Nuevo > Programa**.
3. Seleccione el proyecto o carpeta que contendrá el archivo EGL y, a continuación, seleccione un paquete. Dado que el nombre de programa será idéntico al nombre de archivo, elija un nombre de archivo que se ajuste a los convenios de denominación de componentes de EGL. En el campo Nombre de archivo fuente EGL, teclee el nombre del archivo EGL, por ejemplo myEGLprg. Seleccione un tipo de programa EGL (encontrará los detalles en *Programa básico en formato fuente EGL*, *Programa TextUI en formato fuente EGL*). Si el componente de programa es un programa principal, pulse para quitar la marca de selección de Crear como programa llamado.
4. Pulse en el botón **Finalizar**.

Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13
“Introducción a EGL” en la página 1
“Componente de programa” en la página 139

Tareas relacionadas

“Crear una carpeta fuente de EGL” en la página 127

Consulta relacionada

“Programa básico en formato fuente EGL” en la página 729
“Crear un archivo fuente EGL” en la página 128
“Convenios de denominación” en la página 672
“Programa de UI de texto en formato fuente EGL” en la página 731

Componente de programa

Un *componente de programa* define la unidad lógica central en un programa Java de tiempo de ejecución. Para tener una visión general de los programas principales y llamados así como de los tipos de programa (básico y textUI), consulte la sección *Componentes*.

Cualquier tipo de componente de programa incluye una función llamada *principal*, que representa la lógica que se ejecuta al iniciar el programa. Un programa puede incluir otras funciones y puede acceder a funciones que están fuera del programa. La función *principal* puede invocar estas otras funciones, y cualquier función puede proporcionar control a otros programas.

Las propiedades más importantes de un programa son las siguientes:

- Cada *parámetro* hace referencia a un área de memoria que contiene datos recibidos de un llamador. Los parámetros son globales al programa y sólo son válidos en los programas llamados.
- Cada *variable* hace referencia a un área de memoria que está asignada en el programa y es global al mismo.
- Un *grupo de formularios* es una colección de formularios que presentan datos al usuario:
 - Un programa básico puede presentar datos a una impresora utilizando *formularios de impresión*
 - Un programa de texto puede presentar datos de forma interactiva (utilizando *formularios de texto*) o a una impresora

Para obtener información detallada, consulte la sección *Componente FormGroup*.

- Un *registro de entrada* es un área de memoria global que recibe datos cuando el control se transfiere de forma asíncrona desde otro programa. Un registro de entrada sólo está disponible en un programa principal.
- En programas de texto principales, la propiedad *segmentado* determina qué acciones se realizan automáticamente antes de que el programa emita una sentencia **converse** para presentar un formulario de texto. Para obtener información detallada, consulte la sección *Segmentación*.
- Además, en los programas de texto, un *formulario de entrada* tiene una de las dos finalidades siguientes al iniciar el programa:
 - El formulario se presenta a un usuario que invoca el programa desde un monitor o terminal
 - Como alternativa, los datos que ha introducido un usuario se reciben en el formulario de entrada, que es un área de memoria del propio programa. Esta situación sólo se aplica en el caso de una *conmutación de programa diferido*, que es una transferencia de control de dos pasos producida por una variante de la sentencia **show**:
 1. Un programa envía un formulario de texto al usuario y luego termina
 2. El usuario envía el formulario y, en función de la información del formulario, el envío invoca automáticamente un segundo programa, que contiene el formulario de entrada

Para obtener una lista completa de las propiedades de programa, consulte la sección *Propiedades del componente de programa*.

Conceptos relacionados

“Componente FormGroup” en la página 153

“Componente de función” en la página 140

“Componentes” en la página 17
“Referencias a variables en EGL” en la página 58
“Segmentación en aplicaciones de texto” en la página 160

Tareas relacionadas

“Crear un componente de programa de EGL” en la página 138

Consulta relacionada

“Ayuda de contenido en EGL” en la página 483
“Inicialización de datos” en la página 471
“Formato fuente EGL” en la página 491
“Sentencias EGL” en la página 88
“Componente de programa en formato fuente EGL” en la página 728
“Propiedades de componente de programa” en la página 733

Crear un componente de función de EGL

Un componente de función es una unidad lógica que contiene el primer código de un programa o que se invoca desde otra función. Los componentes de función de EGL están contenidos en archivos de EGL. Para crear un componente de función de EGL, haga lo siguiente:

1. Busque un archivo EGL para que albergue el componente de función y abra el archivo en el editor EGL. Debe crear un archivo EGL si todavía no tiene uno.
2. Teclee las especificaciones del componente de función de acuerdo con la sintaxis de EGL (para conocer los detalles, consulte la sección *Componente de función en formato de código fuente EGL*). Puede utilizar la ayuda de contenido para incluir un esquema de la sintaxis del componente de función en el archivo.
3. Guarde el archivo EGL.

Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13
“Componente de función”

Tareas relacionadas

“Crear un archivo fuente EGL” en la página 128
“Utilizar las plantillas EGL con la ayuda de contenido” en la página 129

Consulta relacionada

“Ayuda de contenido en EGL” en la página 483
“Invocaciones de función” en la página 518
“Componente de función en formato fuente EGL” en la página 527
“Convenios de denominación” en la página 672

Componente de función

Un *componente de función* es una unidad lógica que contiene el primer código del programa o que se invoca desde otra función. La función que contiene el primer código del programa se llama *principal*.

El componente de función puede incluir las siguientes propiedades:

- Un *valor de retorno*, que describe los datos que el componente de función devuelve al llamador
- Un conjunto de *parámetros*, cada uno de los cuales hace referencia a la memoria que asigna y pasa otro componente lógico

- Un conjunto de otras *variables*, cada una de las cuales asigna otra memoria que es local a la función
- Sentencias EGL
- Una especificación acerca de si la función necesita el contexto de programa, tal como se describe en *containerContextDependent*

La función *principal* es poco común ya que no puede devolver un valor ni incluir parámetros y debe declararse dentro de un componente de programa.

Conceptos relacionados

- “Componentes” en la página 17
- “Componente de programa” en la página 139
- “Referencias a variables en EGL” en la página 58
- “Soporte de SQL” en la página 229

Consulta relacionada

- “containerContextDependent” en la página 465
- “Inicialización de datos” en la página 471
- “Formato fuente EGL” en la página 491
- “Invocaciones de función” en la página 518
- “Componente de función en formato fuente EGL” en la página 527
- “Sentencias EGL” en la página 88

Crear un componente de biblioteca de EGL

Un componente de biblioteca de EGL contiene un conjunto de funciones, variables y constantes que pueden utilizar los programas, los PageHandlers u otras bibliotecas. Para crear un componente de biblioteca de EGL, haga lo siguiente:

1. Identifique un proyecto o carpeta para que contenga el archivo. Debe crear un proyecto o carpeta si no tiene uno todavía.
2. En el entorno de trabajo, pulse en **Archivo > Nuevo > Biblioteca**.
3. Seleccione el proyecto o carpeta que contendrá el archivo EGL y, a continuación, seleccione un paquete. Dado que el nombre de biblioteca será idéntico al nombre de archivo, elija un nombre de archivo que se ajuste a los convenios de denominación de componentes de EGL. En el campo Nombre de archivo fuente EGL, teclee el nombre del archivo EGL, por ejemplo myLibrary.
4. Seleccione el tipo de biblioteca pulsando uno de los botones de selección siguientes:
 - **Básica - Crear una biblioteca básica**
 - **Nativa - Crear una biblioteca nativa**
5. Pulse en el botón **Finalizar**.

Conceptos relacionados

- “Proyectos, paquetes y archivos EGL” en la página 13
- “Introducción a EGL” en la página 1
- “Componente de biblioteca de tipo basicLibrary” en la página 142
- “Componente de biblioteca de tipo basicLibrary” en la página 142

Tareas relacionadas

- “Crear una carpeta fuente de EGL” en la página 127
- “Crear un proyecto Web EGL” en la página 126

Consulta relacionada

- “Crear un archivo fuente EGL” en la página 128

“Componente de biblioteca en formato fuente EGL” en la página 649

“Convenios de denominación” en la página 672

Componente de biblioteca de tipo `basicLibrary`

Un *componente de biblioteca* de tipo `basicLibrary` contiene un conjunto de funciones, variables y constantes que pueden ser utilizadas por programas, `PageHandlers` y otras bibliotecas. Se recomienda utilizar bibliotecas para maximizar la reutilización de código y valores comunes.

La especificación de tipo *basicLibrary* indica que el componente se genera en una unidad compilable e incluye valores de EGL y código para la ejecución local. Este tipo es el valor por omisión cuando no se especifica la palabra clave **type**. Para obtener detalles acerca de cómo crear una biblioteca para acceder a una DLL nativa desde un programa Java generado por EGL, consulte la sección *Componente de biblioteca de tipo nativeLibrary*.

Las reglas para una biblioteca de tipo *basicLibrary* son las siguientes:

- Se puede hacer referencia a las funciones, variables y constantes de una biblioteca sin especificar el nombre de biblioteca, pero sólo si la biblioteca se incluye en una declaración de uso específica del programa.
- Las funciones de biblioteca pueden acceder a cualquier variable del sistema que esté asociada al `PageHandler` o programa invocador. Se aplican las siguientes normas:
 - Cuando una función de una biblioteca recibe un registro como un argumento, el registro no puede utilizarse para entrada o salida (E/S) ni para probar un estado de E/S como por ejemplo `endOfFile`. Sin embargo, el código que invoca la biblioteca, puede utilizar el registro de ambas maneras.
 - Cuando declara un registro en una biblioteca, las funciones basadas en biblioteca pueden utilizar el registro para entrada o salida (E/S) y para probar el estado de E/S (para el final del archivo, por ejemplo). Sin embargo, el código que invoca la biblioteca, no puede utilizar el registro de ninguna de las dos maneras.
- Las funciones de biblioteca pueden utilizar cualquier sentencia excepto las siguientes:
 - `converse`
 - `forward`
 - `show`
 - `transfer`
- Una biblioteca no puede acceder a un formulario de texto.
- Una biblioteca que accede a un formulario de impresión debe incluir una declaración de uso para el grupo de formularios relacionado.
- Puede utilizar el modificador **private** en una declaración de función, variable o constante para evitar que el elemento se utilice fuera de la biblioteca.
- Las funciones de biblioteca que están declaradas como públicas (que es el valor por omisión) están disponibles fuera de la biblioteca y no pueden tener parámetros de tipo `loose`, que es una clase especial de tipo primitivo que sólo está disponible si se desea que el parámetro acepte un rango de longitudes de argumento. Para obtener información detallada sobre el tipo `loose`, consulte la sección *Componente de función en formato fuente EGL*.

La biblioteca se genera separadamente de los componentes que la utilizan. El tiempo de ejecución de EGL accede al componente de biblioteca utilizando el valor de la propiedad de biblioteca **alias**, que por omisión toma el valor del nombre de la biblioteca EGL.

Durante la ejecución, la biblioteca se carga cuando se utiliza por primera vez y se descarga cuando el programa o el PageHandler que accedió a la biblioteca sale de la memoria, como ocurre cuando la unidad de ejecución finaliza..

Un PageHandler recibe una nueva copia de la biblioteca siempre que se carga el PageHandler. Además, una biblioteca invocada por otra biblioteca permanece en memoria mientras lo haga la biblioteca invocadora.

Una biblioteca que sólo se utiliza para sus constantes no se carga durante la ejecución ya que las constantes se generan como literales en los programas y PageHandlers que hacen referencia a las mismas.

Conceptos relacionados

“forward” en la página 583

“Componente de función en formato fuente EGL” en la página 527

“Componente de biblioteca en formato fuente EGL” en la página 649

“Componente de biblioteca de tipo basicLibrary” en la página 142

“Unidad de ejecución” en la página 742

“Segmentación en aplicaciones de texto” en la página 160

“show” en la página 646

“transfer” en la página 646

“Declaración use” en la página 954

Consulta relacionada

“converse” en la página 570

“forward” en la página 583

“Componente de función en formato fuente EGL” en la página 527

“Componente de biblioteca en formato fuente EGL” en la página 649

“Unidad de ejecución” en la página 742

“Segmentación en aplicaciones de texto” en la página 160

“show” en la página 646

“transfer” en la página 646

“Declaración use” en la página 954

Componente de biblioteca de tipo nativeLibrary

Una biblioteca de tipo nativeLibrary permite al código Java generado por EGL invocar una sola DLL que esté ejecutándose localmente. El código para esa DLL no está escrito en el lenguaje EGL. Para obtener información sobre el desarrollo de una biblioteca básica que contiene funciones compartidas y valores que *se* escriben en lenguaje EGL, consulte la sección *Componente de biblioteca de tipo basicLibrary*.

En una biblioteca de tipo nativeLibrary, el objetivo de cada función consiste en proporcionar una interfaz a una función DLL. NO puede definir sentencias en la función EGL ni puede declarar variables o constantes en ninguna parte de la biblioteca.

El tiempo de ejecución de EGL accede a una función basada en DLL utilizando el valor de la propiedad de función EGL **alias** que por omisión es el nombre de

función EGL. Establezca esa propiedad explícitamente si el nombre de la función basada en DLL no se ajusta a los convenios descrito en la sección *Convenios de denominación*.

La propiedad de biblioteca **callingConvention** especifica cómo pasa el tiempo de ejecución de EGL entre las dos clases de código:

- El código EGL que invoca la función de biblioteca
- La función de la DLL.

El único valor disponible para **callingConvention** es *I4GL*:

- Los datos se pasan de acuerdo con el formato de pila Informix. Cada parámetro de entrada se coloca en una pila de entrada y cada parámetro de salida se coloca en una pila de salida.
- No puede pasar argumentos como registros o diccionarios. Solo son válidos estos:
 - Las variables primitivas, incluyendo variables de tipo ANY
 - Campos que están en dataTables, formularios de impresión, formularios de texto y registros fijos pero solo si el campo no tiene una subestructura

Los parámetros de las funciones de biblioteca deben ser variables primitivas y pueden ser de tipo ANY, pero no pueden ser de tipo suelto ni pueden incluir el modificador **field**.

La propiedad de biblioteca **dllName** especifica el nombre de DLL, que es final; no puede sobrescribirse durante el despliegue. Si no especifica un valor para la propiedad de biblioteca **dllName**, debe especificar el nombre de DLL en la propiedad de tiempo de ejecución Java `vgj.defaultI4GLNativeLibrary`. Solo hay una propiedad de tiempo de ejecución Java tal para una unidad de ejecución, de modo que solo puede especificarse una DLL aparte de las DLL que se identifican en las bibliotecas EGL.

Tanto si especifica el nombre de la DLL durante el desarrollo (en **dllName**) o durante el despliegue (en `vgj.defaultI4GLNativeLibrary`), la DLL debe residir en la vía de acceso de directorios identificada en una variable de tiempo de ejecución; esa variable es `PATH` (en Windows 2000/NT/XP) o `LIBPATH` (en plataformas UNIX).

Las funciones de biblioteca se declaran automáticamente como públicas para asegurarse de que están disponibles fuera de la biblioteca. En el otro código EGL, puede hacer referencia a una función solamente mediante el nombre de alias de funciones correspondientes, sin especificar el nombre de biblioteca, pero solo si incluye la biblioteca en una declaración de uso específica del programa.

La biblioteca EGL se genera como una clase Java separada del código que accede a la biblioteca y de la DLL. El tiempo de ejecución de EGL accede a esa clase utilizando el valor de la propiedad de biblioteca **alias**, que por omisión toma el valor del nombre de la biblioteca EGL. Establezca esa propiedad explícitamente si el nombre del componente de biblioteca no se ajusta a los convenios de Java.

En tiempo de ejecución, una DLL se carga cuando se utiliza por primera vez y se descarga cuando el programa de acceso o el PageHandler sale de la memoria, tal como ocurre cuando finaliza la unidad de ejecución.

Un PageHandler recibe una nueva copia de la DLL siempre que se carga el PageHandler. Además, una DLL invocada por una biblioteca EGL de tipo basicLibrary permanece en la memoria mientras lo haga la biblioteca invocadora.

La biblioteca nativa siguiente proporciona acceso a una DLL escrita en C:

```
Library myLibrary type nativeLibrary
{callingConvention="I4GL", dllname="mydll"}

Function entryPoint1( p1 int nullable in,
                     p2 date in, p3 time in,
                     p4 interval in, p5 any out)
end

Function entryPoint2( p1 float in,
                     p2 String in,
                     p3 smallint out)
end

Function entryPoint3( p1 any in,
                     p2 any in,
                     p3 any out,
                     p4 CLOB inout)
end
end
```

Conceptos relacionados

“Propiedades de tiempo de ejecución Java” en la página 347
“Componente de biblioteca de tipo basicLibrary” en la página 142

Consulta relacionada

“Componente de función en formato fuente EGL” en la página 527
“Propiedades de ejecución de Java (detalles)” en la página 540
“Componente de biblioteca en formato fuente EGL” en la página 649
“Convenios de denominación” en la página 672
“Unidad de ejecución” en la página 742
“Declaración use” en la página 954

Crear un componente dataTable de EGL

Un componente dataTable de EGL asocia una estructura de datos con una matriz de valores iniciales para la estructura. Para crear un componente dataTable de EGL, haga lo siguiente:

1. Identifique un proyecto o carpeta para que contenga el archivo. Debe crear un proyecto o carpeta si no tiene uno todavía.
2. En el entorno de trabajo, pulse en **Archivo > Nuevo > Tabla de datos**.
3. Seleccione el proyecto o carpeta que contendrá el archivo EGL y, a continuación, seleccione un paquete. Dado que el nombre de dataTable será idéntico al nombre de archivo, elija un nombre de archivo que se ajuste a los convenios de denominación de componentes de EGL. En el campo Nombre de archivo fuente EGL, teclee el nombre del archivo EGL, por ejemplo myDataTable. Seleccione un subtipo de dataTable (encontrará los detalles en *Componente de tabla de datos en formato fuente EGL*).
4. Pulse en el botón **Finalizar**.

Conceptos relacionados

“DataTable” en la página 146
“Proyectos, paquetes y archivos EGL” en la página 13
“Introducción a EGL” en la página 1

Tareas relacionadas

“Crear una carpeta fuente de EGL” en la página 127

“Crear un proyecto Web EGL” en la página 126

Consulta relacionada

“Crear un archivo fuente EGL” en la página 128

“Componente DataTable en formato fuente EGL” en la página 473

“Convenios de denominación” en la página 672

DataTable

Un *dataTable* de EGL consta principalmente de los siguientes componentes:

- Una estructura, en que cada elemento de nivel superior define una columna.
- Una matriz de valores que son coherentes con estas columnas. Cada elemento de esta matriz define una fila.

Por ejemplo, un componente *dataTable* de mensajes de error podría incluir los siguientes componentes:

- La declaración de un campo numérico y un campo de caracteres
- Una lista de valores emparejados, como los siguientes:

```
001 Error 1
002 Error 2
003 Error 3
```

Un *dataTable* no se declara como se declara un elemento de registro o de datos. En cambio, cualquier código que pueda acceder a un *dataTable* puede tratar dicho componente como una variable. Para obtener información detallada sobre el acceso de componentes, consulte la sección *Referencias a componentes*.

Cualquier código que pueda acceder a un *dataTable* tiene la opción de hacer referencia al nombre de componente en una declaración de uso.

Tipos de dataTable

Algunos tipos de un *dataTable* son para la validación en tiempo de ejecución; concretamente, para contener datos para compararlos con la entrada de formulario. (Relacione el un *dataTable* con el campo de entrada cuando declare el componente de formulario). Existen tres tipos de *dataTable* de validación:

matchValidTable

La entrada del usuario debe coincidir con un valor de la primera columna del *dataTable*.

matchInvalidTable

La entrada del usuario debe ser distinta de cualquier valor de la primera columna del *dataTable*.

rangeChkTable

La entrada del usuario debe coincidir con un valor que está comprendido entre los valores de la primera y segunda columna de al menos una fila del *dataTable*. (Los valores inicial y final del rango están incluidos; la entrada del usuario es válida si coincide con un valor de la primera o segunda columna de cualquier fila).

Los otros tipos de *dataTable* son los siguientes:

msgTable

Contiene mensajes de tiempo de ejecución.

basicTable

Contiene otra información que se utiliza en la lógica del programa; por ejemplo, una lista de países y códigos relacionados.

Generación de dataTable

La salida de la generación de dataTable es un par de archivos, cada uno de los cuales tiene un nombre para el dataTable. Un archivo tiene la extensión .java y el otro tiene la extensión .tab. El compilador Java no procesa el archivo .tab, pero éste se incluye en la raíz de la estructura de directorios que contiene el paquete. Si, por ejemplo, el paquete es *my.product.package*, la estructura de directorios es *my/product/package* y el archivo .tab se encuentra en el directorio que contiene el subdirectorio *my*.

No es necesario generar ningún dataTable si se está generando en un paquete en el que previamente se habían generado los mismos dataTable.

Para ahorrar tiempo de generación cuando no necesite generar dataTable, asigne NO a la opción del descriptor de construcción **genTables**.

Propiedades de los dataTable

Puede establecer las siguientes propiedades:

- Se incorpora un **alias** en los nombres de la salida generada. Si no se especifica un alias, en su lugar se utiliza el nombre de componente .
- La propiedad **shared** indica si varios usuarios pueden acceder al dataTable. El valor por omisión es *no*.
- La propiedad **resident** indica si el un dataTable permanece en memoria incluso cuando ningún programa utiliza el un dataTable. (El programa entra en la memoria cuando se accede al mismo por primera vez). El valor predeterminado es *no*. Puede especificar *sí* sólo si la especificación compartida es también *sí*.

Conceptos relacionados

“Referencias a componentes” en la página 21

Consulta relacionada

“Componente DataTable en formato fuente EGL” en la página 473

“Declaración use” en la página 954

Insertar fragmentos de código en archivos EGL y JSP

La vista Fragmentos de código permite reutilizar objetos de programación reutilizables en el código. La vista Fragmentos de código contiene varios trozos de código EGL, así como código de otras muchas tecnologías. Puede utilizar los fragmentos de código proporcionados o añadir sus propios fragmentos a la vista Fragmentos de código. Para obtener más información acerca de cómo utilizar la vista Fragmentos de código, consulte el apartado *Vista Fragmentos de código*.

Para insertar un fragmento de código EGL en el código, haga lo siguiente:

1. Abra el archivo al que desea añadir un fragmento de código.
2. Abra la vista Fragmentos de código.
 - a. Pulse **Ventana > Mostrar vista > Otras**.
 - b. Expanda **Básica** y pulse **Fragmentos de código**.
 - c. Pulse en **Aceptar**.
3. En la vista Fragmentos de código, expanda la bandeja **EGL**. Esta bandeja contiene los fragmentos de código EGL disponibles.
4. Utilice uno de estos métodos para insertar un fragmento de código en el archivo:
 - Pulse y arrastre un fragmento de código en el código fuente.
 - Efectúe una doble pulsación sobre un fragmento de código para insertarlo en la posición del cursor actual. Verá una ventana que describe las variables y los valores del fragmento de código. Si es así, pulse **Insertar**.

Nota: Si el cursor se convierte en un círculo tachado, lo que indica que no se puede insertar el fragmento de código en ese punto, es posible que esté intentando insertar el fragmento de código en el lugar equivocado. Compruebe los detalles del fragmento de código para averiguar si debe insertarse en el código.

5. Cambie los nombres predefinidos de las funciones, variables y componentes de datos del fragmento de código según convenga. La mayoría de fragmentos de código incluyen comentarios que explican qué nombres deben cambiarse.

A continuación se proporcionan los fragmentos de código disponibles en EGL:

Tabla 6. Fragmentos de código disponibles en EGL

Nombre de fragmento de código	Descripción
setCursorFocus	Una función JavaScript que establece el foco del cursor en un campo de formulario especificado en una página Web.
autoRedirect	Una función JavaScript que prueba la presencia de una variable de sesión. Si la variable de sesión no está presente, reenvía el navegador a una página distinta.
getClickedRowValue	Una función EGL que recupera el valor hiperenlazado de una fila en una tabla de datos.

Tabla 6. Fragmentos de código disponibles en EGL (continuación)

Nombre de fragmento de código	Descripción
databaseUpdate	Una función EGL que actualiza una sola fila de una tabla relacional cuando se pasa un registro de un PageHandler.

Conceptos relacionados

Vista Fragmentos de código

Tareas relacionadas

- “Utilizar las plantillas EGL con la ayuda de contenido” en la página 129
- “Establecer el foco en un campo de formulario”
- “Probar navegadores para una variable de sesión”
- “Recuperar el valor de una fila pulsada en una tabla de datos” en la página 151
- “Actualizar una fila en una tabla relacional” en la página 152

Consulta relacionada

Establecer el foco en un campo de formulario

El fragmento de código `setCursorFocus` de la bandeja JSP de la vista Fragmentos de código es una función JavaScript que establece el foco del cursor en un campo de formulario especificado en una página Web. Debe colocarse en un código `<script>` en una página JSP. Para insertar y configurar este fragmento de código, siga estas instrucciones:

1. Inserte el código del fragmento en el código fuente de la página. Para obtener más información, consulte el apartado *Insertar fragmentos de código EGL*.
2. Sustituya `[n]` por el número del campo de formulario que recibirá el foco. Por ejemplo, utilice `[3]` para establecer el foco en el cuarto campo de la página.
3. Establezca el nombre de formulario en `form1`.
4. Cambie el código `<body>` de la página JSP por `<body onload="setfocus();">`.

El código insertado por este fragmento de código es el siguiente:

```
function setFocus() {
    document.getElementById('form1').elements[n].select();
    document.getElementById('form1').elements[n].focus();
}
```

Tareas relacionadas

“Insertar fragmentos de código en archivos EGL y JSP” en la página 149

Probar navegadores para una variable de sesión

El fragmento de código `autoRedirect` de la bandeja JSP de la vista Fragmentos de código, comprueba la presencia de una variable de sesión. Si la variable de sesión no está presente, reenvía el navegador a una página distinta. Este fragmento de código debe colocarse en el código `<head>` de una página JSP después del código `<pageEncoding>`. Para insertar y configurar este fragmento de código, siga estas instrucciones:

1. Inserte el código del fragmento en el código `<head>` de la página, después del código `<pageEncoding>`. Para obtener más información, consulte el apartado *Insertar fragmentos de código EGL*.

2. Sustituya {SessionAttribute} por el nombre de la variable de sesión que se está probando.
3. Sustituya {ApplicationName} por el nombre del proyecto u aplicación.
4. Sustituya {PageName} por el nombre de la página a la que se redirigirá el navegador si la variable de sesión está ausente.

El código insertado por este fragmento de código es el siguiente:

```
<%
if ((session.getAttribute("userID") == null ))
{
String redirectURL =
"http://localhost:9080/EGLWeb/faces/Login.jsp";
response.sendRedirect(redirectURL);
}
%>
```

Tareas relacionadas

“Insertar fragmentos de código en archivos EGL y JSP” en la página 149

Recuperar el valor de una fila pulsada en una tabla de datos

El fragmento de código `getClickedRowValue` de la bandeja EGL de la vista Fragmentos de código es una función que recupera el valor hiperenlazado de una fila pulsada en una tabla de datos. Este fragmento de código debe colocarse en un `PageHandler`. Este fragmento de código tiene los requisitos previos siguientes:

1. La página JSP tiene una tabla de datos.
2. No se ha cambiado el nombre predeterminado de los identificadores JSP.
3. La página está definida como petición en ámbito en `faces-config.xml`, no como sesión.

Para insertar y configurar este fragmento de código, siga estas instrucciones:

1. Inserte el código del fragmento en el `PageHandler`. Para obtener más información, consulte el apartado *Insertar fragmentos de código EGL*.
2. Defina una variable de caracteres o de serie para que reciba el valor pulsado.
3. Añada un hiperenlace de mandato (de la bandeja Componentes Faces en la vista Paleta) a un campo en la tabla de datos.
4. Para el destino del hiperenlace de mandatos, especifique el nombre de la página JSP. El hiperenlace enlaza con su propia página.
5. Añada un parámetro al hiperenlace y déle a ese parámetro el mismo nombre que tiene la variable del `PageHandler` que recibe el valor pulsado.
6. Establezca la propiedad `action` (ubicada en la pestaña Todas de la vista Propiedades) en la función `getVal()`.

El código insertado por este fragmento de código es el siguiente:

```
function getVal()
javaLib.store((objId)"context",
"javax.faces.context.FacesContext",
"getCurrentInstance");
javaLib.store((objId)"root",
(objId)"context", "getViewRoot");
javaLib.store((objId)"parm",
(objId)"root",
"findComponent",
```

```

    "form1:table1:param1");
    recVar = javaLib.invoke((objId)"parm",
                           "getValue");
end

```

Tareas relacionadas

“Insertar fragmentos de código en archivos EGL y JSP” en la página 149

Actualizar una fila en una tabla relacional

El fragmento de código `databaseUpdate` de la bandeja EGL de la vista `Fragmentos de código` es una función que actualiza una sola fila de una tabla relacional cuando se pasa un registro de un `PageHandler`. Este fragmento debe colocarse en una biblioteca EGL. Para insertar y configurar este fragmento de código, siga estas instrucciones:

1. Inserte el código del fragmento en el `PageHandler`. Para obtener más información, consulte el apartado *Insertar fragmentos de código EGL*.
2. Sustituya `{tableName}` y `{keyColumn}` por el nombre de la tabla y la columna de clave primaria correspondiente.

El código insertado por este fragmento de código es el siguiente:

```

Function updateRec(${TableName}New ${TableName})

    // Nombre de función - llamar a esta función
    // pasando el registro ${TableName} como parámetro
    ${TableName}Old ${TableName};

    // Una copia del registro, utilizada
    // para bloquear la fila de la tabla y obtener
    // los valores de fila existentes antes de la actualización
    ${TableName}Old.${KeyColumn} =
        ${TableName}New.${KeyColumn};
    get ${TableName}Old forUpdate;

    // Obtener la fila existente.
    // Advierta que si desea realizar un proceso personalizado
    // debe insertar el código después de esta llamada
    move ${TableName}New to ${TableName}Old byName;

    //Mover los valores actualizados a la fila de copia
    replace ${TableName}Old;

    //Y sustituir la fila en la base de datos.
    sysLib.commit();

    //Comprometer los cambios en la base de datos
    onException
        //Si la actualización falla...
        sysLib.rollback();

    // cancelar todas las actualizaciones de base de datos
    // (suponiendo que esto esté permitido
    // por la base de datos) y llamar a
    // una rutina de manejo de errores personalizada
end
end

```

Tareas relacionadas

“Insertar fragmentos de código en archivos EGL y JSP” en la página 149

Trabajar con formularios de texto e impresión

Crear un componente formGroup de EGL

Un componente formGroup de EGL define una colección de formularios de texto e impresión. Para crear un componente formGroup de EGL, haga lo siguiente:

1. Identifique un proyecto o carpeta para que contenga el archivo. Debe crear un proyecto o carpeta si no tiene uno todavía.
2. En el entorno de trabajo, pulse en **Archivo > Nuevo > Grupo de formularios**.
3. Seleccione el proyecto o carpeta que contendrá el archivo EGL y, a continuación, seleccione un paquete. Dado que el nombre de formGroup será idéntico al nombre de archivo, elija un nombre de archivo que se ajuste a los convenios de denominación de componentes de EGL. En el campo Nombre de archivo fuente EGL, teclee el nombre del archivo EGL, por ejemplo myFormGroup.
4. Pulse en el botón **Finalizar**.

Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13

“Visión general del editor de formularios de EGL” en la página 164

“Editar grupos de formularios con el editor de formularios de EGL” en la página 164

“Componente FormGroup”

“Introducción a EGL” en la página 1

Tareas relacionadas

“Crear una carpeta fuente de EGL” en la página 127

“Crear un proyecto Web EGL” en la página 126

Consulta relacionada

“Crear un archivo fuente EGL” en la página 128

“Componente de formulario en formato fuente EGL” en la página 511

“Convenios de denominación” en la página 672

Componente FormGroup

Un componente EGL FormGroup tiene dos finalidades:

- Define una colección de formularios de texto e impresión. (Los formularios que son exclusivos del componente se definen dentro del componente o se incluyen mediante una declaración de uso. Los formularios que son comunes a varios componentes FormGroup se incluyen mediante una declaración de uso.)
- Define de cero a muchas *áreas flotantes*, como se describe en la sección *Componente de formulario*

Un grupo de formularios no se declara como se declara un elemento de registro o de datos. En su lugar, el programa accede a un componente FormGroup (y a los formularios relacionados) sólo si se cumple lo siguiente:

- La ubicación del componente FormGroup es accesible al programa, como se describe en la sección *Referencias a componentes*
- Una declaración de uso del programa hace referencia al componente FormGroup

Un programa no puede incluir más de dos componentes formGroup; y si se especifican dos, uno debe ser un *grupo de ayuda*. Un grupo de ayuda contiene uno

o más *formularios de ayuda*, que son formularios de sólo lectura que proporcionan información como respuesta a una pulsación de usuario.

Los formularios sólo están disponibles durante la ejecución si se genera el componente FormGroup. La salida generada es una clase para el componente FormGroup y una clase para cada componente de formulario.

Durante la preparación, cada una de estas entidades se procesa en un módulo de carga de ejecución distinto. El entorno de ejecución EGL maneja la interacción del programa generado y el código específico del formulario.

Los componentes de formulario no pueden generarse por separado.

Conceptos relacionados

“Componente de formulario”

“Referencias a componentes” en la página 21

“Editar grupos de formularios con el editor de formularios de EGL” en la página 164

Consulta relacionada

“Declaración use” en la página 954

Componente de formulario

Un *componente de formulario* es una unidad de presentación. Describe el diseño y las características de un conjunto de campos que se muestran simultáneamente al usuario.

Un formulario no se declara como se declara un elemento de registro o de datos. Para acceder a un componente de formulario, el programa debe incluir una declaración de uso que haga referencia al grupo de formularios relacionado.

Un componente de formulario puede ser de dos tipos, *texto* o *impresión*:

- Un formulario de tipo *text* define un diseño que se visualiza en una ventana de mandato. Con una excepción, cualquier formulario de texto puede tener campos de longitud constante y campos de longitud variable, incluidos los campos de longitud variable que aceptan la entrada de usuario. La excepción es un *formulario de ayuda*, que se utiliza exclusivamente para presentar información de longitud constante.
- Un formulario de tipo *impresión* define un diseño que se envía a una impresora. Cualquier formulario de impresión puede tener campos de longitud constante y de longitud variable.

Las propiedades del formulario determinan el tamaño y la posición de la salida en una pantalla o página y especifican las características de formato de dicha salida.

Un formulario cualquiera puede visualizarse en uno o más *dispositivos*, cada uno de los cuales es un periférico de salida o es el equivalente operativo de un periférico de salida.

- Un *dispositivo de pantalla* es un terminal, un monitor o un emulador de terminal. La superficie de salida es una pantalla.
- Un *dispositivo de impresión* es un archivo que puede enviarse a una impresora o es la propia impresora. La superficie de salida es una página.

Tanto si es de tipo *texto* o *impresión*, un formulario también puede clasificarse de las siguientes maneras:

- Un *formulario fijo* tiene una fila y una columna iniciales específicas en relación con la superficie de salida del dispositivo. Por ejemplo, se puede asignar un formulario de impresión fijo que empiece en la línea 10, columna 1 de una página.
- Un *formulario flotante* no tiene ninguna fila o columna inicial específica; en su lugar, un formulario flotante se coloca en la siguiente línea no ocupada de una subárea de la superficie de salida que se declare. La subárea declarada se denomina *área flotante*.

Por ejemplo, puede declarar un área flotante para que sea un rectángulo que empieza en la línea línea 10, llega hasta la línea 20 y tiene el ancho máximo del dispositivo de salida. Si tiene un formulario flotante de una sola línea del mismo ancho, puede construir un bucle que actúe de la manera siguiente cada 20 veces:

1. Coloca datos en la correlación flotante
2. Escribe la correlación flotante en la línea siguiente del área flotante

Una o más áreas flotante se declaran en el componente FormGroup, pero sólo una puede aceptar formularios flotantes para un determinado dispositivo. Si intenta presentar un formulario flotante en ausencia de un área flotante, toda la superficie de salida se trata como un área flotante.

- Un *formulario parcial* es más pequeño que el tamaño estándar de la superficie de salida de un determinado dispositivo. Puede declarar y situar formularios parciales de modo que varios formularios se visualicen en diferentes posiciones horizontales. Aunque se pueden especificar las columnas inicial y final de un formulario parcial, no se pueden visualizar los formularios que están juntos uno al lado de otro.

Los detalles adicionales son específicos del tipo de formulario:

- Formularios de impresión
- Formularios de texto

Conceptos relacionados

“Formularios de impresión” en la página 156

“Formularios de texto” en la página 158

“Editar grupos de formularios con el editor de formularios de EGL” en la página 164

“Plantillas de formulario en el editor de formularios de EGL” en la página 171

Tareas relacionadas

“Crear un formulario en el editor de formularios de EGL” en la página 166

Consulta relacionada

“Componente FormGroup en formato fuente EGL” en la página 508

“Componente de formulario en formato fuente EGL” en la página 511

Crear un formulario de impresión de EGL

Un formulario de impresión es un componente de formulario de EGL que define un diseño para enviarlo a una impresora. Para crear un formulario de impresión de EGL, haga lo siguiente:

1. Busque un archivo de EGL para que albergue el formulario de impresión y abra el archivo en el editor de EGL. Debe crear un archivo EGL si todavía no tiene uno.
2. Teclee las especificaciones del formulario de impresión de acuerdo con la sintaxis de EGL (para conocer los detalles, consulte la sección *Componente de*

formulario en formato de código fuente de EGL). Puede utilizar la ayuda de contenido para incluir un esquema de la sintaxis del componente de formulario en el archivo.

3. Guarde el archivo EGL.

Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13

“Componente de formulario” en la página 154

“Formularios de impresión”

“Editar grupos de formularios con el editor de formularios de EGL” en la página 164

Tareas relacionadas

“Crear un archivo fuente EGL” en la página 128

“Utilizar las plantillas EGL con la ayuda de contenido” en la página 129

“Crear un formulario en el editor de formularios de EGL” en la página 166

Consulta relacionada

“Ayuda de contenido en EGL” en la página 483

“Componente de formulario en formato fuente EGL” en la página 511

“Convenios de denominación” en la página 672

Formularios de impresión

Los formularios y sus tipos se explican en la sección *Componente de formulario*. La página actual describe cómo deben presentarse los formularios de impresión.

Proceso de impresión: La impresión es un proceso que consta de dos pasos:

- En primer lugar, codifique las sentencias **print**, cada una de las cuales añade un formulario a un almacenamiento intermedio de tiempo de ejecución
- A continuación, el entorno de ejecución EGL añade los símbolos necesarios para iniciar una página nueva, envía todos los formularios colocados en el almacenamiento intermedio a un dispositivo de impresión y borra el contenido del almacenamiento intermedio. Estos servicios se proporcionan como respuesta a alguna de las siguientes circunstancias:
 - El programa ejecuta una sentencia **close** en un formulario de impresión que tiene como destino el mismo dispositivo de impresión; o bien
 - El programa está en modalidad segmentada (como se describe en la sección *Segmentación*) y ejecuta una sentencia **converse**; o bien
 - El programa se ha llamado utilizando un programa no EGL (y no VisualAge Generator) y el programa llamado finaliza; o bien
 - El programa principal de la unidad de ejecución finaliza.

En el caso de una salida de varios formularios, las sentencias **print** deben invocarse en el orden en que se desean presentar los formularios. Considere el siguiente ejemplo:

- Al principio de la salida, un formulario fijo identifica una empresa que realiza una compra y un número de pedido
- En un área flotante subsiguiente, una serie de formularios flotantes formateados de forma idéntica identifican cada elemento del pedido de la empresa
- Al final de la salida, un formulario fijo indica el número de pantallas o páginas necesarias para desplazarse por la lista de elementos

Puede conseguir esta salida enviando una serie de sentencias **print**, cada una de las cuales actúa sobre un formulario de impresión. Estas sentencias hacen referencia a los formularios *en el siguiente orden*:

1. Formulario superior
2. Formulario flotante, presentado mediante una sentencia **print** que se invoca repetidamente en un bucle
3. Formulario inferior

Los símbolos necesarios para iniciar una página nueva se insertan en diversas circunstancias, pero puede hacer que se inserten invocando la función de sistema `ConverseLib.pageEject` antes de emitir una sentencia **print**.

Consideraciones acerca de los formularios fijos: Hay que tener en cuenta lo siguiente respecto a los formularios fijos:

- Si emite una sentencia **print** para un formulario fijo que tiene una línea inicial mayor que la línea actual, EGL inserta los símbolos necesarios para avanzar el dispositivo de impresión hasta la línea especificada. De forma similar, si emite una sentencia **print** para un formulario fijo que tiene una línea inicial menor que la línea actual, EGL inserta los símbolos necesarios para iniciar una página nueva.
- Si un formulario fijo recubre *algunas pero no todas* las líneas de otro formulario fijo, EGL inserta automáticamente los símbolos necesarios para iniciar una página nueva y coloca el segundo formulario fijo en la página nueva.
- Si un formulario fijo recubre *todas* las líneas de otro formulario fijo, EGL sustituye el formulario existente sin borrar el resto de la salida del almacenamiento intermedio. Para mantener la salida existente y colocar el nuevo formulario en la página siguiente, invoque la función de sistema `ConverseLib.pageEject` antes de emitir la sentencia **print** para el nuevo formulario.

Consideraciones acerca de los formularios flotantes: Se pueden producir las siguientes equivocaciones al utilizar formularios flotantes:

- Se emite una sentencia **print** para colocar un formulario flotante más allá del final del área flotante; o bien
- Se emite una sentencia **print** que recubre al menos parcialmente un área flotante con un formulario fijo y luego se emite una sentencia **print** para añadir un formulario flotante al área flotante.

En cualquiera de los dos casos, el resultado es que EGL inserta los símbolos necesarios para iniciar una página nueva, y el formulario flotante se coloca en la primera línea del área flotante de la página nueva. Si, por ejemplo, la página es similar a la salida de pedido y elemento descrita anteriormente, la página nueva no incluye el formulario fijo superior.

Destino de impresión: Cuando EGL procesa una sentencia **close** para presentar un archivo de impresión, la salida se envía a una impresora o conjunto de datos. Puede especificar el destino en cualquiera de los tres momentos siguientes:

- Durante la prueba (como se describe en *Depurador EGL*)
- Durante la generación (como se describe en *Asociaciones de recursos y tipos de archivo*)
- Durante la ejecución (como se describe en relación con la variable de sistema `ConverseVar.printerAssociation`)

Conceptos relacionados

“Depurador EGL” en la página 279

“Componente FormGroup en formato fuente EGL” en la página 508

“Componente de formulario en formato fuente EGL” en la página 511

“Componente de formulario” en la página 154
“Asociaciones de recursos y tipos de archivo” en la página 304
“Segmentación en aplicaciones de texto” en la página 160

Consulta relacionada

“pageEject()” en la página 790
“printerAssociation” en la página 923

Crear un formulario de texto de EGL

Un formulario de texto es un formulario de EGL que define un diseño para visualizarlo en una ventana de mandatos. Para crear un formulario de texto de EGL, haga lo siguiente:

1. Busque un archivo de EGL para que albergue el formulario de texto y abra el archivo en el editor de EGL. Debe crear un archivo EGL si todavía no tiene uno.
2. Teclee las especificaciones del formulario de texto de acuerdo con la sintaxis de EGL (para conocer los detalles, consulte la sección *Componente de formulario en formato de código fuente de EGL*). Puede utilizar la ayuda de contenido para incluir un esquema de la sintaxis del componente de formulario en el archivo.
3. Guarde el archivo EGL.

Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13
“Componente de formulario” en la página 154
“Formularios de texto”
“Editar grupos de formularios con el editor de formularios de EGL” en la página 164

Tareas relacionadas

“Crear un archivo fuente EGL” en la página 128
“Crear un formulario en el editor de formularios de EGL” en la página 166
“Utilizar las plantillas EGL con la ayuda de contenido” en la página 129

Consulta relacionada

“Ayuda de contenido en EGL” en la página 483
“Componente de formulario en formato fuente EGL” en la página 511
“Convenios de denominación” en la página 672

Formularios de texto

Los formularios y sus tipos se explican en la sección *Componente de formulario*. La página actual describe cómo se deben presentar los formularios de texto.

La sentencia **converse** es suficiente para proporcionar al usuario acceso a un único formulario de texto fijo. El flujo lógico del programa sólo continúa después de que el usuario responde al formulario visualizado. También se puede construir la salida a partir de varios formularios, como en el siguiente caso:

- Al principio de la salida, un formulario fijo identifica una empresa que realiza una compra y un número de pedido
- En un área flotante subsiguiente, una serie de formularios flotantes formateados de forma idéntica identifican cada elemento del pedido de la empresa
- Al final de la salida, un formulario fijo indica el número de pantallas necesarias para desplazarse por la lista de elementos

Son necesarios dos pasos:

1. En primer lugar, construya la salida de pedido y elemento codificando una serie de sentencias **display**, cada una de las cuales añade un formulario a un almacenamiento intermedio de tiempo de ejecución pero no presenta los datos en pantalla. Cada sentencia **display** actúa sobre uno de los siguientes formularios:
 - Formulario superior
 - Formulario flotante, presentado mediante una sentencia **display** que se invoca repetidamente en un bucle
 - Formulario inferior
2. A continuación, el entorno de ejecución EGL presenta todos los formularios de texto colocados en almacenamiento intermedio al dispositivo de salida como respuesta a una de estas dos situaciones:
 - El programa ejecuta una sentencia **converse**; o bien
 - El programa finaliza.

En la mayoría de casos, el último formulario de la salida de pantalla se presenta codificando una sentencia **converse** en lugar de una sentencia **display**.

Cada formulario fijo tiene una posición en pantalla, de modo que no importa el orden en que se especifican los formularios, en relación entre sí y en relación con la visualización repetida de los formularios flotantes. El contenido del almacenamiento intermedio se borra cuando la salida se envía a la pantalla.

Si se recubre un formulario de texto con otro, no se produce ningún error, pero se aplica lo siguiente:

- Si un formulario parcial recubre algunas líneas de otro formulario fijo, EGL sustituye el formulario existente sin borrar el resto de la salida del almacenamiento intermedio. Si desea borrar la salida existente antes de visualizar el nuevo formulario, invoque la función de sistema `ConverseLib.clearScreen` antes de emitir la sentencia **display** o **converse** para el nuevo formulario.
- Si utiliza una sentencia **display** o **converse** para colocar una correlación flotante más allá de la parte inferior del área flotante, se borran todos los formularios flotantes de este área flotante y el formulario añadido se coloca en la primera línea de la misma área flotante.
- Si un formulario flotante recubre un formulario fijo, se aplica lo siguiente:
 - El formulario flotante sólo sobrescribe las líneas del formulario fijo que están en el área flotante
 - Si una línea del formulario flotante que incluye un campo de longitud variable sobrescribe una línea del formulario fijo, el resultado es imprevisible.

Tanto si se presenta un formulario como si se presentan muchos, el destino de la salida es el dispositivo de pantalla en el que el usuario empezó la unidad de ejecución.

Conceptos relacionados

“Componente de formulario” en la página 154

Consulta relacionada

“Componente de formulario en formato fuente EGL” en la página 511

“Componente FormGroup en formato fuente EGL” en la página 508

“clearScreen()” en la página 789

Segmentación en aplicaciones de texto

La segmentación se refiere a la forma en que un programa interactúa con su entorno antes de emitir una sentencia **converse**.

Por omisión, un programa que presenta formularios de texto está *no segmentado*, lo que significa que el programa se comporta como si estuviera siempre en memoria y proporcionando un servicio sólo a un usuario. Se aplican las siguientes normas antes de que un programa no segmentado emita una sentencia **converse**:

- No se comprometen las bases de datos y otros recursos recuperables
- No se liberan los bloqueos
- Se conservan las posiciones de archivo y de base de datos
- Las tablas EGL de un solo usuario no se renuevan; sus valores son los mismos antes y después de emitirse la sentencia converse
- De forma parecida, no se renuevan las variables de sistema

Un programa llamado siempre está no segmentado

Un programa no segmentado puede ser más fácil de codificar. Por ejemplo, no es necesario volver a adquirir un bloqueo en una fila SQL después de una sentencia **converse**. Un inconveniente es que las filas SQL están retenidas mientras el usuario está pensando, un comportamiento que causa problemas de rendimiento para otros usuarios que necesitan acceder a la misma fila SQL.

Existen dos técnicas para liberar o renovar los recursos antes de que se emita una sentencia converse en un programa no segmentado:

- Puede establecer la variable de sistema **ConverseVar.commitOnConverse** en 1. Los resultados son los siguientes antes de una sentencia converse:

- Se comprometen las bases de datos y otros recursos recuperables
- Se liberan los bloqueos
- No se conservan las posiciones de archivo y de base de datos

El valor de **ConverseVar.commitOnConverse** no afecta nunca a las variables de sistema ni a las tablas EGL.

- Una segunda técnica para manejar la sentencia converse consiste en establecer la propiedad *segmentado* del programa de texto en *sí*, ya sea cambiando una propiedad de programa durante el desarrollo o bien estableciendo la variable de sistema **ConverseVar.segmentedMode** en 1 durante la ejecución. La segmentación produce los siguientes resultados antes de que se emita una sentencia converse:

- Se comprometen las bases de datos y otros recursos recuperables
- Se liberan los bloqueos
- No se conservan las posiciones de archivo y de base de datos
- Se renuevan las tablas EGL de un solo usuario; sus valores siguen siendo los mismos que cuando empezó el programa
- Se renuevan las variables de sistema; sus valores siguen siendo los mismos que cuando empezó el programa, excepto para un subconjunto de variables cuyos valores se guardan *a lo largo de los segmentos*

El comportamiento de un programa segmentado no se ve afectado por el valor de la variable de sistema **ConverseVar.commitOnConverse**.

Conceptos relacionados

“Componente de programa” en la página 139

Código de datos modificados y propiedad **modified**

Cada elemento de un formulario de texto tiene un *código de datos modificados*, que es un valor de estado que indica si se considera que el usuario ha cambiado el elemento de formulario cuando el formulario se presentó por última vez.

Como se describe más adelante, el código de datos modificados de un elemento es distinto de la propiedad **modified**, que está establecida en el programa y que preestablece el valor del código de datos modificados.

Interacción con el usuario: En la mayoría de casos, el código de datos modificados está preestablecido en *no* cuando el programa presenta el formulario al usuario; a continuación, si el usuario cambia los datos en el elemento de formulario, el código de datos modificados se establece en *sí*, y la lógica del programa puede realizar lo siguiente:

- Utilizar una función o tabla de datos para validar los datos modificados (como ocurre automáticamente cuando el código de datos modificados del elemento es *sí*)
- Detectar que el usuario ha modificado el elemento (por ejemplo, utilizando una sentencia condicional del tipo *if item modified*)

El usuario establece el código de datos modificados escribiendo un carácter en el elemento o bien suprimiendo un carácter. El código de datos modificados permanece establecido, incluso si el usuario, antes de enviar el formulario, devuelve el contenido del campo al valor que se ha presentado.

Cuando vuelve a visualizarse un formulario debido a un error, el formulario sigue procesando la misma sentencia converse. Como resultado, los campos que se modificaron en converse tienen el código de datos modificado establecido en *sí* al volver a visualizarse el formulario. Por ejemplo, si se entran datos en un campo que tiene una función de validador, la función puede invocar a la función `ConverseLib.validationFailed` para establecer un mensaje de error y provocar que vuelva a visualizarse el formulario. En este caso, cuando se pulse una tecla de acción, la función de validador volverá a ejecutarse ya que el código de datos modificado del campo sigue establecido en *sí*.

Establecer la propiedad **modified:** Es posible que desee que el programa realice una tarea independientemente de si el usuario ha modificado un determinado campo; por ejemplo:

- Es posible que desee forzar la validación de un campo de contraseña aunque el usuario no haya introducido datos en dicho campo
- Puede especificar una función de validación para un campo de gran importancia (incluso para un campo protegido) de modo que el programa realice siempre una determinada *validación entre campos*, lo que significa que la lógica del programa valida un grupo de campos y considera cómo el valor de un campo afecta a la validez de otro.

Para manejar los casos anteriores, puede establecer la propiedad **modified** de un determinado elemento en la lógica del programa o en la declaración de formulario:

- En la lógica que precede a la presentación del formulario, incluya una sentencia del tipo *set item modified*. El resultado es que cuando se presenta el formulario, el código de datos modificados del elemento está preestablecido en *sí*.
- En la declaración de formulario, establezca la propiedad **modified** del elemento en *sí*. En este caso, se aplican las siguientes normas:

- Cuando el formulario se presenta por primera vez, el código de datos modificados del elemento está preestablecido en *sí*.
- Si se produce alguna de las siguientes situaciones antes de que se presente el formulario, el código de datos modificados está preestablecido en *sí* cuando se presenta el formulario:
 - El código ejecuta una sentencia del tipo *set item initial*, que vuelve a asignar el contenido y los valores de propiedad originales del elemento; o bien
 - El código ejecuta una sentencia del tipo *set item initialAttributes*, que vuelve a asignar los valores de propiedad originales (pero no el contenido) de cada elemento del formulario; o bien
 - El código ejecuta una sentencia del tipo *set form initial*, que vuelve a asignar el contenido y los valores de propiedad originales de cada elemento del formulario; o bien
 - El código ejecuta una sentencia del tipo *set form initialAttributes*, que vuelve a asignar los valores de propiedad originales (pero no el contenido) de cada elemento del formulario

Las sentencias *set* afectan al valor de la propiedad **modified**, no al valor actual del código de datos modificados. Una prueba del tipo *if item modified* se basa en el valor del código de datos modificados que estaba en vigor cuando los datos de formulario se devolvieron por última vez al programa. Si intenta probar el código de datos modificados de un elemento antes de que la lógica presente el formulario *por primera vez*, se produce un error durante la ejecución.

En caso de que necesite detectar si el usuario (y no el programa) ha modificado un elemento, asegúrese de que el valor del código de datos modificados del elemento esté preestablecido en *no*:

- Si la propiedad **modified** del elemento está establecida en *no* en la declaración de formulario, no utilice una sentencia del tipo *set item modified*. En ausencia de esta sentencia, la propiedad **modified** se establece automáticamente en *no* antes de cada presentación de formulario.
- Si la propiedad **modified** del elemento está establecida en *sí* en la declaración de formulario, utilice una sentencia del tipo *set item normal* en la lógica que precede a la presentación de formulario. Esta sentencia establece la propiedad **modified** en *no* y (como resultado secundario) presenta el elemento como no protegido, con intensidad normal.

Probar si el formulario está modificado: Se considera que el formulario en su conjunto está modificado si el código de datos modificados está establecido en *sí* para alguno de los elementos variables de formulario. Si prueba el estado de modificación de un formulario que todavía no se ha presentado al usuario, el resultado de la prueba es FALSE.

Ejemplos: Supongamos los siguientes valores en el formulario *form01*:

- La propiedad **modified** para el campo *item01* está establecida en *no*
- La propiedad **modified** para el campo *item02* está establecida en *sí*

La lógica siguiente muestra el resultado de varias pruebas:

```
// el resultado de la prueba es false porque no se ha
// ejecutado una sentencia converse para el formulario
if (form01 is modified)
;
end

// produce un error de entorno de ejecución porque no se ha
```

```

// ejecutado una sentencia converse para el formulario
if (item01 is modified)
;
end

// supongamos que el usuario modifica ambos elementos
converse form01;

// el resultado de la prueba es true
if (item01 is modified)
;
end

// el resultado de la prueba es true
if (item02 is modified)
;
end

// establece la propiedad modified en no
// en la siguiente sentencia converse para el formulario
set item01 initialAttributes;

// establece la propiedad modified en sí
// en la siguiente sentencia converse para el formulario
set item02 initialAttributes;

// el resultado de la prueba es true
// (la sentencia establecida anteriormente sólo se aplica
// en la siguiente sentencia converse para el formulario
if (item01 is modified)
;
end

// supongamos que el usuario no modifica ninguno de los dos elementos
converse form01;

// el resultado de la prueba es false porque el programa ha establecido
// el código de datos modificados en no y el usuario no ha introducido datos
if (item01 is modified)
;
end

// el resultado de la prueba es true porque el programa ha establecido
// el código de datos modificados en sí
if (item02 is modified)
;
end

// supongamos que el usuario no modifica ninguno de los dos elementos
converse form01;

// el resultado de la prueba es false
if (item01 is modified)
;
end

// el resultado de la prueba es false porque la presentación
// no era la primera y el programa no ha restablecido las
// propiedades de elemento en sus valores iniciales
if (item02 is modified)
;
end

```

Visión general del editor de formularios de EGL

El editor de formularios de EGL le permite editar gráficamente un componente formGroup. El editor de formularios trabaja con componentes formGroup, sus componentes de formulario y los campos de dichos componentes de formulario de forma muy parecida a como otros editores gráficos trabajan con archivos como, por ejemplo, páginas Web y diagramas Web.

El editor de formularios tiene los siguientes componentes:

- El propio editor, que muestra la representación gráfica del grupo de formularios y el código fuente de dicho grupo de formularios. Puede conmutar entre la representación gráfica y el código fuente pulsando las pestañas **Diseño** y **Fuente** que se encuentran en la parte inferior del editor. Los cambios realizados en la vista Fuente o en la vista Diseño se reflejan inmediatamente en la otra vista.
- La vista Propiedades, que muestra las propiedades de EGL del formulario o campo que actualmente está seleccionado en el editor.
- La vista Paleta, que muestra los tipos de formularios y campos que pueden crearse en el editor.
- La vista Esquema, que muestra una vista jerárquica del grupo de formularios abierto en el editor.

Para obtener más información sobre cómo utilizar el editor de formularios, consulte el apartado *Editar grupos de formularios con el editor de formularios de EGL*.

Conceptos relacionados

“Componente FormGroup” en la página 153

“Componente de formulario” en la página 154

“Opciones de visualización del editor de formularios de EGL” en la página 174

“Filtros de formulario en el editor de formularios de EGL” en la página 176

“Editar grupos de formularios con el editor de formularios de EGL”

Tareas relacionadas

“Crear un formulario en el editor de formularios de EGL” en la página 166

“Establecer preferencias para el editor de formularios de EGL” en la página 175

“Establecer preferencias de entrada paleta editor formularios EGL” en la página 170

Editar grupos de formularios con el editor de formularios de EGL

El editor de formularios de EGL le permite editar gráficamente un componente formGroup. El editor de formularios trabaja con componentes formGroup, sus componentes de formulario y los campos de dichos componentes de formulario de forma muy parecida a como otros editores gráficos trabajan con archivos como, por ejemplo, páginas Web y diagramas Web. En cualquier momento, puede pulsar la pestaña **Fuente** que se encuentra en la parte inferior del editor y ver el código fuente EGL que el editor está generando. El editor de formularios tiene las siguientes características:

- El editor de formularios puede editar el tamaño y las propiedades de un grupo de formularios. Para editar las propiedades de un grupo de formularios, abra el grupo de formularios en el editor de formularios y cambie sus propiedades en la vista Propiedades. Para redimensionar un grupo de formularios, ábralo en el editor de formularios y elija un tamaño en caracteres en la lista de la parte superior del editor.

- El editor de formularios puede crear, editar y suprimir formularios de un grupo de formularios. Para crear un formulario, pulse en el tipo apropiado de formulario de la vista Paleta y arrastre un rectángulo que represente el tamaño y la ubicación del formulario en el editor. Para editar un formulario, pulse en el mismo para seleccionarlo y luego utilice la vista Propiedades para editar sus propiedades. También puede arrastrar un formulario para moverlo o redimensionarlo utilizando los handles de redimensionado que aparecen en el borde de un formulario seleccionado. Muchas de estas funciones están disponibles cuando se pulsa con el botón derecho del ratón en un formulario para abrir su menú emergente. Consulte el apartado *Crear un formulario en el editor de formularios de EGL*.
- El editor de formularios utiliza plantillas para crear tipos de formularios de uso común, como por ejemplo formularios emergentes y menús emergentes. Estos formularios tienen bordes, secciones y campos preestablecidos. Consulte el apartado *Plantillas de formulario en el editor de formularios de EGL*.
- El editor de formularios puede crear, editar y suprimir campos de un formulario. Para crear un campo, pulse en el tipo apropiado de campo de la vista Paleta y arrastre un rectángulo que represente el tamaño y la ubicación del campo en el editor. Sólo puede añadir un campo dentro de un formulario existente. Para editar un campo, pulse en el mismo para seleccionarlo y luego utilice la vista Propiedades para editar sus propiedades. También puede arrastrar un campo para moverlo o redimensionarlo utilizando los handles de redimensionado que aparecen en el borde de un formulario seleccionado. Muchas de estas funciones están disponibles cuando se pulsa con el botón derecho del ratón en un campo para abrir su menú emergente. Consulte el apartado *Crear un campo de longitud constante* o el apartado *Crear un campo de longitud variable*.
- Los filtros pueden impedir que los formularios se muestren en el editor de formularios, permitiéndole imitar el aspecto del grupo de formularios durante la ejecución. Para conmutar entre filtros, crear un filtro o editar filtros, utilice el botón **Filtros** que se encuentra en el parte superior del editor. Consulte el apartado *Filtros de formulario en el editor de formularios de EGL* o el apartado *Crear un filtro*.
- Puede personalizar el aspecto del editor de formularios utilizando las opciones de visualización que se encuentran en la parte superior del editor y estableciendo las preferencias del editor en la ventana Preferencias. Por ejemplo, estas opciones pueden visualizar una cuadrícula sobre el grupo de formularios, aumentar o disminuir el nivel de zoom y mostrar u ocultar valores de ejemplo en los campos. Consulte el apartado *Opciones de visualización del editor de formularios de EGL* o el apartado *Establecer preferencias para el editor de formularios de EGL*.

Conceptos relacionados

“Visión general del editor de formularios de EGL” en la página 164

“Componente FormGroup” en la página 153

“Componente de formulario” en la página 154

“Opciones de visualización del editor de formularios de EGL” en la página 174

“Filtros de formulario en el editor de formularios de EGL” en la página 176

“Plantillas de formulario en el editor de formularios de EGL” en la página 171

Tareas relacionadas

“Crear un filtro” en la página 166

“Crear un formulario emergente” en la página 171

“Crear un menú emergente” en la página 172

“Visualizar un registro en un formulario de texto o impresión” en la página 173

- “Establecer preferencias para el editor de formularios de EGL” en la página 175
- “Establecer preferencias de entrada paleta editor formularios EGL” en la página 170
- “Crear un formulario en el editor de formularios de EGL”
- “Crear un campo de longitud constante” en la página 167
- “Crear un campo de longitud variable en un formulario de texto o impresión” en la página 168

Consulta relacionada

- “Componente FormGroup en formato fuente EGL” en la página 508
- “Componente de formulario en formato fuente EGL” en la página 511

Crear un filtro

Para crear un filtro nuevo en el editor de formularios de EGL, siga estos pasos:

1. Abra un grupo de formularios en el editor de formularios.
2. En el editor de formularios, pulse el botón **Filtros**. Se abre la ventana Filtros.
3. En la vista Filtros, pulse el botón **Nuevo**. Se abre el diálogo Filtro nuevo.
4. En el diálogo Filtro nuevo, escriba un nombre para el filtro y pulse **Aceptar**.
5. Seleccione los formularios que desea visualizar mientras el filtro está activo llevando a cabo uno o más de los pasos siguientes:
 - Desmarque los recuadros de selección que están junto a los formularios que desea que oculte el filtro.
 - Marque los recuadros de selección que están junto a los formularios que desea que muestre el filtro.
 - Pulse el botón **Seleccionar todos** para mostrar todos los formularios.
 - Pulse el botón **Deseleccionar todos** para ocultar todos los formularios.
6. Pulse **Aceptar**.

El filtro nuevo ahora está activo. Puede conmutar entre filtros utilizando la lista que está junto al botón **Filtros**.

Conceptos relacionados

- “Visión general del editor de formularios de EGL” en la página 164
- “Editar grupos de formularios con el editor de formularios de EGL” en la página 164
- “Opciones de visualización del editor de formularios de EGL” en la página 174
- “Filtros de formulario en el editor de formularios de EGL” en la página 176

Tareas relacionadas

- “Crear un formulario en el editor de formularios de EGL”

Crear un formulario en el editor de formularios de EGL

Para crear un formulario en el editor de formularios de EGL, siga estos pasos:

1. Abra un grupo de formularios en el editor de formularios.
2. En la vista Paleta, pulse **Formulario de texto** o **Formulario de impresión**.
3. En el grupo de formularios del editor, pulse y arrastre un rectángulo que indique el tamaño y la forma del formulario. Se abre la ventana Crear componente de formulario.
4. En la ventana Crear componente de formulario, escriba un nombre para el formulario en el campo **Especificar nombre de componente**. Este nombre será el nombre del componente de formulario en el código fuente EGL.
5. Pulse en **Aceptar**.
6. Pulse en el formulario y edite sus propiedades en la vista Propiedades.

- Añada campos al formulario según sea necesario. Consulte los apartados *Crear un campo de longitud constante* y *Crear un campo de longitud variable*.

También puede crear formularios a partir de las plantillas de la vista Paleta. Estas plantillas crean formularios con aspectos y campos predefinidos. Consulte el apartado *Crear un formulario emergente* o *Crear un menú emergente*.

Conceptos relacionados

- “Visión general del editor de formularios de EGL” en la página 164
- “Editar grupos de formularios con el editor de formularios de EGL” en la página 164
- “Componente FormGroup” en la página 153
- “Componente de formulario” en la página 154
- “Opciones de visualización del editor de formularios de EGL” en la página 174
- “Filtros de formulario en el editor de formularios de EGL” en la página 176
- “Plantillas de formulario en el editor de formularios de EGL” en la página 171

Tareas relacionadas

- “Crear un filtro” en la página 166
- “Crear un formulario emergente” en la página 171
- “Crear un menú emergente” en la página 172
- “Visualizar un registro en un formulario de texto o impresión” en la página 173
- “Crear un campo de longitud constante”
- “Crear un campo de longitud variable en un formulario de texto o impresión” en la página 168

Consulta relacionada

- “Componente FormGroup en formato fuente EGL” en la página 508
- “Componente de formulario en formato fuente EGL” en la página 511

Crear un campo de longitud constante

Los campos de longitud constante muestran una serie de texto que no cambia en un formulario. A diferencia de los campos de longitud variable, el código EGL no puede acceder a los campos de longitud constante. Para insertar un campo de longitud constante en un formulario, siga estos pasos:

- Abra un grupo de formularios en el editor de formularios de EGL.
- Si el grupo de formularios no tiene ningún formulario, añada un formulario al grupo de formularios. Consulte el apartado *Crear un formulario*.
- En la vista Paleta, pulse en un tipo de campo de longitud constante para añadir. De forma predeterminada, los siguientes tipos de campos de longitud constante están disponibles:

Tabla 7. Campos de longitud constante disponibles en la vista Paleta

Nombre de campo	Color predeterminado	Intensidad predeterminada	Resaltado predeterminado	Protección por omisión
Título	Azul	Negrita	Ninguno	Saltar
Cabecera de columna	Azul	Negrita	Ninguno	Saltar
Etiqueta	Cian	Normal	Ninguno	Saltar
Instrucciones	Cian	Normal	Ninguno	Saltar
Ayuda	Blanco	Normal	Ninguno	Saltar

Estos campos son ejemplos de campos de texto de longitud constante utilizados habitualmente en una interfaz basada en texto. Puede personalizar los campos individuales después de colocarlos en un formulario. También puede

personalizar el color, la intensidad y el resaltado predeterminados de los campos disponibles en la vista Paleta. Consulte el apartado *Establecer preferencias para las entradas de la paleta del editor de formularios de EGL*.

4. En un formulario del editor, pulse y mantenga pulsado el ratón para dibujar un rectángulo que represente el tamaño y la ubicación del campo. Un recuadro de vista previa que está junto al cursor del ratón muestra el tamaño del campo así como su ubicación respecto al formulario.

Nota: Sólo puede añadir un campo dentro de un formulario existente.

5. Cuando el campo tenga el tamaño correcto, suelte el ratón. Se crea el campo nuevo.
6. Escriba el texto que desea que se visualice en el campo.
7. En la vista Propiedades, establezca las propiedades del campo nuevo.

Conceptos relacionados

“Visión general del editor de formularios de EGL” en la página 164

“Editar grupos de formularios con el editor de formularios de EGL” en la página 164

“Componente de formulario” en la página 154

Tareas relacionadas

“Establecer preferencias de entrada paleta editor formularios EGL” en la página 170

“Crear un campo de longitud variable en un formulario de texto o impresión”

Consulta relacionada

“Componente de formulario en formato fuente EGL” en la página 511

Crear un campo de longitud variable en un formulario de texto o impresión

Los campos de longitud variable pueden servir como texto de entrada o salida de un formulario. Cada campo de longitud variable se basa en un tipo primitivo EGL o en un componente `DataItem`. A diferencia de los campos de longitud constante, el código EGL puede acceder a los campos de longitud variable. Para insertar un campo de longitud variable en un formulario, siga estos pasos:

1. Abra un grupo de formularios en el editor de formularios de EGL.
2. Si el grupo de formularios no tiene ningún formulario, añada un formulario al grupo de formularios. Consulte el apartado *Crear un formulario*.
3. En la vista Paleta, pulse en un tipo de campo de longitud variable para añadir. De forma predeterminada, los siguientes tipos de campos de longitud variable están disponibles:

Tabla 8. Campos de longitud variable disponibles en la vista Paleta

Nombre de campo	Color predeterminado	Intensidad predeterminada	Resaltado predeterminado	Protección por omisión
Entrada	Verde	Normal	Subrayado	No
Salida	Verde	Normal	Ninguno	Saltar
Mensaje	Rojo	Negrita	Ninguno	Saltar
Contraseña	Verde	Invisible	Ninguno	No

Estos campos son ejemplos de campos de texto de longitud variable utilizados habitualmente en una interfaz basada en texto. Puede personalizar los campos individuales después de colocarlos en un formulario. También puede personalizar el color, la intensidad y el resaltado predeterminados de los

campos disponibles en la vista Paleta. Consulte el apartado *Establecer preferencias para las entradas de la paleta del editor de formularios de EGL*.

4. En un formulario del editor, pulse y mantenga pulsado el ratón para dibujar un rectángulo que represente el tamaño y la ubicación del campo. Un recuadro de vista previa que está junto al cursor del ratón muestra el tamaño del campo así como su ubicación respecto al formulario.

Nota: Sólo puede añadir un campo dentro de un formulario existente.

5. Cuando el campo tenga el tamaño correcto, suelte el ratón. Se abre la ventana Campo de EGL nuevo.
6. En la ventana Campo de datos EGL nuevo, especifique el nombre del campo nuevo en el campo **Nombre**.
7. Realice una de las siguientes acciones para seleccionar el tipo de campo:
 - Para utilizar un tipo primitivo, pulse en un tipo primitivo en la lista **Tipo**.
 - Para utilizar un componente DataItem, siga estos pasos:
 - a. Pulse **dataItem** en la lista **Tipo**. Se abre la ventana Seleccionar un componente DataItem.
 - b. En la ventana Seleccionar un componente DataItem, pulse en un componente DataItem de la lista o escriba el nombre de un componente.
 - c. Pulse en **Aceptar**.
8. Según sea necesario, escriba valores en el campo o campos **Dimensiones** para establecer las dimensiones del campo de variable nuevo.
9. Si desea convertir el campo en una matriz, marque el recuadro de selección **Matriz**.
10. Si el recuadro de selección **Matriz** está marcado, pulse **Siguiente** y continúe con los pasos siguientes. En caso contrario, pulse **Finalizar** y no continúe con estos pasos. Se crea el campo nuevo y no es necesario continuar con el resto de los pasos ya que éstos sólo son aplicables si se crea una matriz.
11. En la página Propiedades de matriz de la ventana Campo de EGL nuevo, escriba el tamaño de la matriz en el campo **Tamaño de matriz**.
12. Para la orientación, elija **Vertical** u **Horizontal** en los botones **Orientación de índice**.
13. En **Diseño**, especifique el número de campos verticales y horizontales en los campos **Campos verticales** y **Campos horizontales**.
14. En **Espacios**, especifique la cantidad de espacio entre las filas y columnas de la matriz en los campos **Líneas entre filas** y **Espacios entre columnas**.
15. Pulse en **Finalizar**. Se crea el campo nuevo en el grupo de formularios.

Una vez que haya creado el campo nuevo, pulse en el campo para seleccionarlo y establecer las propiedades del mismo en la vista Propiedades.

Dado que los campos de longitud variable no tienen ningún valor predeterminado, pueden ser invisibles si no están resaltados. Para marcar cada campo de longitud variable con el texto de ejemplo apropiado, pulse el botón **Conmutar valores de ejemplo** situado en la parte superior del editor.

Una vez que haya creado un campo de longitud variable, puede pulsar dos veces en el mismo en el editor para abrir la ventana Editar propiedades de tipo. Desde esta ventana, puede editar el campo de las siguientes maneras:

- Cambie el nombre del campo escribiendo un nombre nuevo en el campo **Nombre de campo**.

- Seleccione un nuevo tipo de campo en la lista **Tipo**.
- Cambie la precisión del campo especificando un número nuevo en el campo **Precisión**.

Cuando haya terminado de editar las propiedades del campo en la ventana Editar propiedades de tipo, pulse **Aceptar**.

Conceptos relacionados

“Visión general del editor de formularios de EGL” en la página 164

“Editar grupos de formularios con el editor de formularios de EGL” en la página 164

“Componente de formulario” en la página 154

Tareas relacionadas

“Establecer preferencias de entrada paleta editor formularios EGL”

“Crear un campo de longitud constante” en la página 167

Consulta relacionada

“Componente de formulario en formato fuente EGL” en la página 511

Establecer preferencias de entrada paleta editor formularios EGL

Las preferencias para la paleta del editor de formularios de EGL controlan el color, intensidad y resaltado predeterminados para los tipos de campos de longitud constante y variable de la paleta. Para cambiar estas preferencias, siga estos pasos:

1. En la barra de menús, pulse **Ventana > Preferencias**. Se abre la ventana Preferencias.
2. En el panel izquierdo de la ventana Preferencias, expanda **EGL > Editor de formularios de EGL** y pulse **Entradas de la paleta de EGL**.
3. En el panel derecho de la ventana Preferencias, para cada tipo de campo de longitud constante y variable de la lista **Entradas de la paleta**, seleccione las siguientes opciones:
 - En la lista **Color**, pulse un color predeterminado para ese tipo de campo.
 - En la lista **Intensidad**, pulse una intensidad predeterminada para ese tipo de campo.
 - En los botones de selección **Resaltado**, pulse un estilo de resaltado predeterminado para ese tipo de campo.
 - En los botones de selección **Protección**, elija si el campo está protegido contra la actualización de usuario por omisión. Para obtener más información acerca de la protección de los campos, consulte el apartado *Propiedades y campos ConsoleField*.

Nota: Puede restaurar todas las entradas de la paleta a sus valores predeterminados pulsando **Restaurar valores predeterminados**.

4. Cuando haya terminado de establecer las preferencias para las entradas de la paleta, pulse **Aceptar**

Conceptos relacionados

“Visión general del editor de formularios de EGL” en la página 164

“Editar grupos de formularios con el editor de formularios de EGL” en la página 164

Tareas relacionadas

“Establecer preferencias para el editor de formularios de EGL” en la página 175

“Crear un campo de longitud constante” en la página 167

“Crear un campo de longitud variable en un formulario de texto o impresión” en la página 168

Consulta relacionada

“Propiedades y campos de ConsoleField” en la página 441

Plantillas de formulario en el editor de formularios de EGL

El editor de formularios de EGL utiliza plantillas para crear tipos de formularios y campos de uso común. Estas plantillas se listan en la bandeja **Plantillas** de la vista Paleta.

El editor de formularios puede crear formularios a partir de las plantillas. Estos formularios tienen bordes, secciones y campos preestablecidos. Para crear un formulario a partir de una plantilla, consulte el apartado *Crear un formulario emergente* o *Crear un menú emergente*.

El editor de formularios puede crear un grupo de campos utilizando un registro EGL como plantilla. Para crear campos que representen datos de un registro EGL, consulte el apartado *Visualizar un registro en un formulario*.

Conceptos relacionados

“Visión general del editor de formularios de EGL” en la página 164

“Editar grupos de formularios con el editor de formularios de EGL” en la página 164

Tareas relacionadas

“Crear un formulario emergente”

“Crear un menú emergente” en la página 172

“Visualizar un registro en un formulario de texto o impresión” en la página 173

“Establecer preferencias para el editor de formularios de EGL” en la página 175

“Establecer preferencias de entrada paleta editor formularios EGL” en la página 170

“Crear un formulario en el editor de formularios de EGL” en la página 166

Crear un formulario emergente

Un formulario emergente es un tipo especial de formulario que puede añadirse a un grupo de formularios. En esencia, un formulario emergente es lo mismo que un formulario de texto corriente, pero los formularios emergentes se crean con características preestablecidas, como por ejemplo bordes y secciones. Para crear un formulario emergente en el editor de formularios de EGL, siga estos pasos:

1. Abra un grupo de formularios en el editor de formularios.
2. En la vista Paleta, pulse **Formulario emergente**.
3. En el grupo de formularios del editor, pulse y arrastre un rectángulo que indique el tamaño y la forma del formulario emergente. Se abre la ventana Crear componente de formulario.
4. En la ventana Crear componente de formulario, escriba un nombre para el formulario en el campo **Especificar nombre de componente**. Este nombre será el nombre del componente de formulario en el código fuente EGL.
5. Pulse **Aceptar**. Se abre la ventana Plantilla de formulario emergente nuevo.
6. En la ventana Plantilla de formulario emergente nuevo, especifique los caracteres que utilizará para los bordes del formulario en los campos **Carácter vertical** y **Carácter horizontal**.
7. Pulse un color para el borde en la lista **Color**.
8. Pulse una intensidad para el borde en la lista **Intensidad**.
9. Pulse un valor de resaltado en los botones de selección **Resaltado**.

10. Repita los siguientes pasos para cada sección que desee añadir al formulario. Debe añadir una sección al formulario como mínimo.
 - a. En **Secciones emergentes**, pulse el botón **Añadir**. Se abre la ventana Crear sección de formulario emergente.
 - b. En la ventana Crear sección de formulario emergente, escriba un nombre para la sección en el campo **Nombre de sección**.
 - c. En el campo **Número de filas**, especifique el número de filas de la sección. No especifique un número superior al número de filas restantes, que se visualiza en la parte inferior de la ventana Plantilla de formulario emergente nuevo.
 - d. Pulse en **Aceptar**.
 - e. Utilice los botones **Arriba** y **Abajo** para establecer el orden de los campos.

Nota: El número total de filas en las secciones del campo emergente no puede ser mayor que el número total de filas en el campo emergente. Al añadir secciones, preste atención en el campo **Filas efectivas restantes** y recuerde que las líneas divisorias requieren una fila adicional para cada campo nuevo.
11. Cuando haya terminado de añadir secciones al campo emergente, pulse **Finalizar**. El nuevo formulario emergente se crea en el editor.
12. Añada campos al formulario según sea necesario. Consulte los apartados *Crear un campo de longitud constante* y *Crear un campo de longitud variable*.

Conceptos relacionados

“Visión general del editor de formularios de EGL” en la página 164

“Editar grupos de formularios con el editor de formularios de EGL” en la página 164

“Plantillas de formulario en el editor de formularios de EGL” en la página 171

Tareas relacionadas

“Establecer preferencias para el editor de formularios de EGL” en la página 175

“Crear un formulario en el editor de formularios de EGL” en la página 166

“Crear un campo de longitud constante” en la página 167

“Crear un campo de longitud variable en un formulario de texto o impresión” en la página 168

“Crear un menú emergente”

Crear un menú emergente

Un menú emergente es un tipo especial de formulario que puede añadirse a un grupo de formularios. En esencia, un menú emergente es lo mismo que un formulario de texto corriente, pero los menús emergentes se crean con características preestablecidas, como por ejemplo un título, texto de ayuda y un número especificado de opciones de menú. Para crear un menú emergente en el editor de formularios de EGL, siga estos pasos:

1. Abra un grupo de formularios en el editor de formularios.
2. En la vista Paleta, pulse **Menú emergente**.
3. En el grupo de formularios del editor, pulse y arrastre un rectángulo que indique el tamaño y la forma del menú emergente. Se abre la ventana Crear componente de formulario.
4. En la ventana Crear componente de formulario, escriba un nombre para el formulario en el campo **Especificar nombre de componente**. Este nombre será el nombre del componente de formulario en el código fuente EGL.
5. Pulse **Aceptar**. Se abre la ventana Plantilla de menú emergente nuevo.

6. En la Plantilla de menú emergente nuevo, especifique el tamaño del menú emergente en los campos **Anchura** y **Altura**. De forma predeterminada, estos campos se pueblan con el tamaño del formulario creado en el editor.
7. En el campo **Título del menú**, escriba el título del menú.
8. En el campo **Número de opciones de menú**, escriba el número de opciones de menú que tendrá el menú emergente.
9. En el campo **Texto de ayuda del menú**, escriba cualquier texto de ayuda adicional para el menú.
10. Pulse en **Finalizar**. El nuevo menú emergente se crea en el editor.
11. Añada campos al nuevo menú emergente y edite los campos existentes según convenga. Consulte los apartados *Crear un campo de longitud constante* y *Crear un campo de longitud variable*.

Conceptos relacionados

“Visión general del editor de formularios de EGL” en la página 164

“Editar grupos de formularios con el editor de formularios de EGL” en la página 164

“Plantillas de formulario en el editor de formularios de EGL” en la página 171

Tareas relacionadas

“Establecer preferencias para el editor de formularios de EGL” en la página 175

“Crear un formulario en el editor de formularios de EGL” en la página 166

“Crear un campo de longitud constante” en la página 167

“Crear un campo de longitud variable en un formulario de texto o impresión” en la página 168

“Crear un formulario emergente” en la página 171

Visualizar un registro en un formulario de texto o impresión

La plantilla **Registro**, que se encuentra en la bandeja **Plantillas** de la vista Paleta, crea un grupo de campos de formulario que son equivalentes a los campos de un componente de registro de EGL. Para crear los campos de formulario, siga estos pasos:

1. Abra un grupo de formularios en el editor de formularios de EGL.
2. Cree un formulario. Consulte el apartado *Crear un formulario en el editor de formularios de EGL*.
3. En la vista Paleta, pulse **Registro**.
4. En un formulario del editor, pulse y mantenga pulsado el ratón para dibujar un rectángulo que represente el tamaño y la ubicación de los campos. Un recuadro de vista previa que está junto al cursor del ratón muestra el tamaño de los campos del registro así como su ubicación respecto al campo.

Nota: Sólo puede añadir un registro dentro de un formulario existente.

5. Cuando el registro tenga el tamaño correcto, suelte el ratón. Se abre la ventana Colocación de registros EGL.
6. En la ventana Colocación de registros EGL, pulse **Examinar**. Se abre el diálogo Seleccionar un componente de registro.
7. En el diálogo Seleccionar un componente de registro, pulse el nombre del componente de registro que desea utilizar o bien escriba el nombre de un componente de registro.
8. Pulse **Aceptar**. Ahora la ventana Crear un registro se puebla con una lista de los campos de dicho registro.
9. Utilizando uno o más de los métodos siguientes, seleccione y organice los campos de componentes de registro que desea visualizar como campos en el formulario:

- Para eliminar un campo, pulse en su nombre y luego pulse **Eliminar**.
 - Para añadir un campo, siga estos pasos:
 - a. Pulse el botón **Añadir**. Se abre la ventana Editar entrada de tabla.
 - b. En la ventana Editar entrada de tabla, escriba un nombre para el campo en el recuadro **Nombre de campo**.
 - c. En la lista **Tipo**, seleccione un tipo para el campo.
 - d. Si es necesario para el tipo que ha seleccionado, especifique la precisión del campo en el campo **Precisión**.
 - e. Especifique la anchura del campo en el campo **Anchura de campo**.
 - f. Si desea que el campo sea un campo de entrada, marque el recuadro de selección **Hacer que este campo sea un campo de entrada**. En caso contrario, desmarque el recuadro de selección.
 - g. Pulse en **Aceptar**.
 - Para editar un campo, siga estos pasos:
 - a. Pulse en el nombre del campo.
 - b. Pulse el botón **Editar**. Se abre la ventana Editar entrada de tabla.
 - c. En la ventana Editar entrada de tabla, escriba un nombre para el campo en el recuadro **Nombre de campo**.
 - d. En la lista **Tipo**, seleccione un tipo para el campo.
 - e. Si es necesario para el tipo que ha seleccionado, especifique la precisión del campo en el campo **Precisión**.
 - f. Especifique la anchura del campo en el campo **Anchura de campo**.
 - g. Si desea que el campo sea un campo de entrada, marque el recuadro de selección **Hacer que este campo sea un campo de entrada**. En caso contrario, desmarque el recuadro de selección.
 - h. Pulse en **Aceptar**.
 - Para subir o bajar campos en la lista, utilice los botones **Arriba** y **Abajo**.
10. Utilizando los botones de selección **Orientación**, elija una orientación vertical u horizontal para los campos.
 11. En el campo **Número de filas**, especifique el número de filas que desea que tenga el grupo de campos.
 12. Si desea que el grupo de campos tenga una fila de cabecera, marque el recuadro de selección **Crear fila de cabecera**.
 13. Pulse en **Finalizar**.

Conceptos relacionados

“Visión general del editor de formularios de EGL” en la página 164

“Editar grupos de formularios con el editor de formularios de EGL” en la página 164

“Plantillas de formulario en el editor de formularios de EGL” en la página 171

Tareas relacionadas

“Crear un formulario en el editor de formularios de EGL” en la página 166

“Crear un campo de longitud constante” en la página 167

“Crear un campo de longitud variable en un formulario de texto o impresión” en la página 168

Opciones de visualización del editor de formularios de EGL

El editor de formularios de EGL tiene opciones de visualización que permiten controlar la forma en que aparecen los grupos de formularios en el editor durante el diseño. Estas opciones no cambian el aspecto del grupo de formularios durante

la ejecución. De izquierda a derecha en la parte superior del editor, los botones que controlan las opciones de visualización son los siguientes:

Conmutar líneas de cuadrícula

Esta opción muestra una cuadrícula sobre el grupo de formularios que ayuda a dimensionar y organizar los formularios. Para cambiar el color de la cuadrícula, consulte el apartado *Establecer preferencias para el editor de formularios de EGL*.

Conmutar valores de ejemplo

Esta opción inserta valores de ejemplo en los campos de longitud variable, que de lo contrario serían invisibles.

Conmutar la modalidad de negro y blanco

Esta opción conmuta el fondo del editor de negro a blanco.

Nivel de zoom

Establece el nivel de ampliación del editor.

Existen otros botones en la parte superior del editor que controlan el tamaño del grupo de formularios y los filtros del editor. Consulte el apartado *Editar grupos de formularios con el editor de formularios de EGL* o el apartado *Filtros de formulario en el editor de formularios de EGL*.

Conceptos relacionados

“Visión general del editor de formularios de EGL” en la página 164

“Filtros de formulario en el editor de formularios de EGL” en la página 176

Tareas relacionadas

“Editar grupos de formularios con el editor de formularios de EGL” en la página 164

“Crear un filtro” en la página 166

“Establecer preferencias para el editor de formularios de EGL”

Establecer preferencias para el editor de formularios de EGL

Las preferencias para el editor de formularios de EGL pueden cambiar el aspecto del editor de formularios, como por ejemplo el color de fondo y el color de cuadrícula. Para cambiar las preferencias del editor de formularios, siga estos pasos:

1. En la barra de menús, pulse **Ventana > Preferencias**. Se abre la ventana Preferencias.
2. En el panel izquierdo de la ventana Preferencias, expanda **EGL** y pulse **Editor de formularios de EGL**.
3. En el panel derecho de la ventana Preferencias, seleccione las preferencias para el editor de formularios:
 - En el campo **Color de fondo**, seleccione un color de fondo para el editor de formularios.
 - En el campo **Color de cuadrícula**, seleccione un color de cuadrícula para el editor de formularios.
 - Si desea mostrar un borde alrededor de los campos, marque el recuadro de selección **Resaltar campos** y seleccione un color.
 - Si desea mostrar reglas en la parte superior izquierda del editor de formularios, marque el recuadro de selección **Mostrar reglas**.
 - En la lista **Font**, pulse un font para los campos y pulse un tamaño en la lista adyacente.

Nota: Elija un fondo monospace para asegurarse de que los campos se visualizan con el tamaño correcto en el editor de formularios. En un font monospace todos los caracteres tienen la misma anchura, como por ejemplo Courier New.

- Si desea que los campos que parpadean se visualicen en cursiva en el editor, marque el recuadro de selección **Mostrar visualmente campos que parpadean**. Esta opción no cambia el aspecto de los campos durante la ejecución; sólo cambia su aspecto durante el diseño.

Nota: Para restaurar la ventana de preferencias del editor de formularios de EGL a los valores predeterminados, pulse **Restaurar valores predeterminados**.

4. Cuando haya terminado de establecer las preferencias para el editor de formularios de EGL, pulse **Aceptar**.

Conceptos relacionados

“Visión general del editor de formularios de EGL” en la página 164

“Editar grupos de formularios con el editor de formularios de EGL” en la página 164

“Filtros de formulario en el editor de formularios de EGL”

Tareas relacionadas

“Establecer preferencias de entrada paleta editor formularios EGL” en la página 170

Filtros de formulario en el editor de formularios de EGL

Los filtros limitan los formularios que se muestran en el editor de formularios de EGL. Puede definir cualquier número de filtros, pero sólo puede haber un filtro activo a la vez. Los filtros sólo afectan a la presentación del grupo de formularios durante el diseño; no afectan en modo alguno al código EGL. Consulte el apartado *Crear un filtro*.

Puede conmutar entre los filtros activos utilizando la lista que está junto al botón **Filtros** en la parte superior del editor. Para crear, editar o suprimir filtros, pulse el botón **Filtros**.

En la ventana Filtros, puede gestionar los filtros utilizando las siguientes funciones:

- Seleccione los filtros en la lista.
- Añada un nuevo filtro pulsando en **Nuevo**.
- Suprima un filtro seleccionándolo en la lista y pulsando **Eliminar**.
- Seleccione qué formularios se visualizan cuando el filtro está activo.

Conceptos relacionados

“Visión general del editor de formularios de EGL” en la página 164

“Editar grupos de formularios con el editor de formularios de EGL” en la página 164

Tareas relacionadas

“Crear un filtro” en la página 166

“Crear un formulario en el editor de formularios de EGL” en la página 166

Crear una interfaz de usuario de consola

Interfaz de usuario de consola

La interfaz de usuario de consola (ConsoleUI) es una tecnología para visualizar datos en un formato basado en texto en una pantalla Windows o UNIX. Esta tecnología sólo está disponible en programas Java generados por EGL, no en PageHandlers.

La interfaz que se crea con ConsoleUI puede visualizarse en Windows 2000/NT/XP o UNIX X-windows, ya sea localmente o a través de una sesión de terminal remoto.

La ConsoleUI es distinta de la interfaz de usuario de texto (TextUI) y las dos no pueden funcionar en el mismo programa:

- Cuando TextUI está en vigor, el estilo de la interfaz es como el que se utiliza en un programa de sistema principal que interactúa con terminales 3270. El programa presenta un formulario de texto pero no procesa la entrada de usuario cuando el usuario se mueve de un campo al siguiente. Cuando el usuario envía el formulario (pulsando la tecla **Intro**, en la mayoría de casos), todos los datos del formulario regresan al programa, y sólo entonces el programa valida los datos; si la validación se realiza satisfactoriamente, el programa ejecuta la siguiente sentencia codificada.
- Cuando ConsoleUI está en vigor, el estilo de la interfaz es como el que se utiliza en un programa basado en UNIX que interactúa con terminales basados en caracteres. El programa presenta un formulario de consola y puede responder inmediatamente a un evento de usuario, como por ejemplo cuando el usuario pulsa la tecla **Tabulador** para mover un cursor en pantalla al campo siguiente. La validación se realiza campo a campo, y se puede restringir el cursor al campo actual hasta que el usuario ha introducido datos válidos en el mismo.

Cuando utilice consoleUI, normalmente codificará un programa de la manera siguiente:

1. Declare un conjunto de variables que se basen en los componentes ConsoleUI, que siempre están disponibles; no defina los componentes que son específicos de ConsoleUI.
2. Abra una entidad visual como, por ejemplo, un formulario incluyendo una variable consoleUI como argumento al invocar la función EGL apropiada. De forma alternativa, puede abrir una entidad visual invocando una función EGL como **displayFormByName**, que acepta un nombre que se conoce durante la ejecución.
3. Haga referencia a la entidad visual de una sentencia EGL **openUI**, que permite la interacción de usuario vinculando eventos concretos (como por ejemplo pulsaciones teclas de usuario) a una determinada lógica.

El usuario de una aplicación consoleUI puede pulsar teclas para interactuar con la visualización en pantalla, pero las pulsaciones de ratón no tienen ningún efecto.

ConsoleUI puede aceptar la entrada de usuario en un campo, pero sólo si se ha especificado un *binding* (enlace), que es una correspondencia entre el campo de entrada y una variable de tipo primitivo. El tiempo de ejecución de EGL actúa de la manera siguiente:

- Utiliza el valor de la variable como contenido inicial de un campo visualizado; y
- Mueve la entrada del usuario a dicha variable en cuanto el usuario sale del campo.

ConsoleUI también permite interactuar con los usuarios en *modalidad de línea*, que es una modalidad de proceso en que el código sólo lee o escribe una línea a la vez. Las implicaciones de la modalidad de línea son las siguientes:

- En el entorno de trabajo de Eclipse, el usuario interactúa con la vista Consola
- En un programa que se ha invocado con un indicador de mandatos, el usuario interactúa con la ventana de mandatos
- En un programa que se ejecuta bajo Curses en UNIX, el usuario interactúa con la ventana donde se visualiza la UI y se suspende la interacción normal basada en ventanas.

ConsoleUI es equivalente a la tecnología de interfaz de usuario del producto Informix 4GL.

Tareas relacionadas

“Crear una interfaz con consoleUI”

Consulta relacionada

“Componentes de ConsoleUI y variables relacionadas” en la página 180

“Opciones de pantalla ConsoleUI para UNIX” en la página 184

“Biblioteca ConsoleLib de EGL” en la página 757

“openUI” en la página 620

“Utilización de new en ConsoleUI” en la página 183

Crear una interfaz con consoleUI

La interfaz de usuario de consola (ConsoleUI) es una tecnología para visualizar datos en un formato basado en texto en una pantalla Windows o UNIX.

Los pasos para crear una interfaz con consoleUI son los siguientes:

1. Cree un archivo fuente EGL
2. Escriba un programa que incluya los elementos de lenguaje descritos en *Componentes de ConsoleUI y variables relacionadas*
3. Genere código Java a partir del archivo fuente EGL
4. Ejecute el archivo Java generado como una aplicación

A continuación se detallan cada una de estas tareas.

Crear un archivo fuente EGL

1. En el entorno de trabajo, en la perspectiva EGL, seleccione **Archivo>Nuevo>Archivo fuente EGL**. O bien, desde cualquier perspectiva, seleccione **Archivo>Nuevo>Otros>Archivo fuente EGL..**
2. En la pantalla del asistente, especifique la información siguiente:
 - **Carpeta fuente:** la ubicación de directorios que albergará el archivo fuente EGL.

- **Paquete:** la ubicación del paquete que albergará el archivo fuente EGL. Este campo es opcional.
 - **Nombre de archivo fuente EGL:** el nombre de archivo del archivo fuente de Interfaz de usuario de consola, como por ejemplo **myConsoleUI**.
3. Seleccione **Finalizar** para crear el archivo. Se añade una extensión (**.egl**) automáticamente al final del nombre de archivo. El archivo EGL aparece en la vista Explorador de proyectos y se abre automáticamente en el editor de EGL por omisión.

Escribir el programa ConsoleUI

Para poblar el archivo fuente y crear ConsoleUI, deberá utilizar los elementos de lenguaje de ConsoleUI, presentados en el tema de ayuda de visión general **egl.ui.console** y definidos a fondo en los temas de ayuda de ConsoleUI **Biblioteca**, **Sentencia OpenUI**, **Tipos de registro** y **Enumeraciones**.

Una aplicación ConsoleUI debe incluir como mínimo los elementos siguientes:

1. PROGRAM...END
2. Función main()
3. Sentencia OpenUI

Nota: Aunque la **sentencia OpenUI** es fundamental para ConsoleUI, puede escribir un programa ConsoleUI satisfactorio sin una **sentencia OpenUI**.

Generar código Java a partir de código fuente EGL

Para generar un archivo Java:

1. En el Editor EGL, pulse el botón derecho del ratón sobre el archivo ConsoleUI. Aparecerá un menú de contexto.
2. Seleccione **Generar**.

Nota: No se puede generar un archivo fuente **.egl** de ConsoleUI para COBOL.

Ejecutar el archivo Java generado como una aplicación

Para ejecutar el archivo Java generado:

1. En el Explorador de proyectos, pulse el botón derecho del ratón sobre el archivo Java generado (**.java**). Aparecerá un menú de contexto.
2. Seleccione **Ejecutar>Ejecutar como>Aplicación Java**.
3. O bien, con el archivo Java abierto en el editor, seleccione **Ejecutar>Ejecutar como>Aplicación Java** en el menú principal.
4. ConsoleUI se visualizará en una ventana.

Una aplicación ConsoleUI puede visualizarse en una sesión de terminal "curses-based" o una ventana gráfica basada en Swing. Los usuarios de UNIX tienen una opción de pantalla más flexible que se describe en el tema de ayuda *Opciones de pantalla de ConsoleUI para UNIX*.

Nota: IBM no soporta la utilización de ConsoleUI y TextUI en el mismo programa.

Conceptos relacionados

"Interfaz de usuario de consola" en la página 177

Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

“Componentes de ConsoleUI y variables relacionadas”

“Opciones de pantalla ConsoleUI para UNIX” en la página 184

“openUI” en la página 620

Componentes de ConsoleUI y variables relacionadas

Cuando se trabaja con consoleUI, se crean las siguientes clases de variables, las cuales se basan en los componentes de consoleUI relacionados:

- Window
- Prompt
- ConsoleField
- ConsoleForm
- Menu
- MenuItem

La biblioteca **ConsoleLib** también incluye variables de sistema de tipo `PresentationAttributes`. Las variables de sistema controlan los aspectos visuales de la salida visualizada; y para cambiar aspectos de la pantalla, puede cambiar dichas variables estableciendo los campos **color**, **highlight** e **intensity** de `PresentationAttributes`. Para obtener información detallada sobre estos campos, consulte el apartado *Campos de PresentationAttributes en consoleUI de EGL*.

Window

Una ventana es un área rectangular donde puede colocar otras entidades visuales que se representan como variables.

Cuando se visualiza una ventana y no hay otras ventanas activas, la nueva ventana está dentro de la *ventana de pantalla*, que es un rectángulo que tiene las características básicas de cualquier ventana del sistema operativo. Para UNIX se aplica una diferencia cuando la biblioteca `Curses` está utilizándose; en este caso, la visualización de una ventana de consoleUI coloca la ventana de terminal existente en modalidad de ventanas.

Cualquier ventana adicional que visualice aparece en la parte de contenido de la ventana de pantalla, normalmente en la parte superior de la ventana que ya ha abierto. También es posible que las ventanas se visualicen una junto a la otra.

Al declarar una ventana, puede establecer varias propiedades. **Position**, por ejemplo, es la ubicación respecto a la esquina superior izquierda de la pantalla; y **size** es la altura y anchura de la ventana en número de caracteres.

A continuación se proporciona un ejemplo de declaración de ventana:

```
myWindow WINDOW
{name="myWindow", position = [2,2],
 size = [18,75], color = red, hasborder=yes};
```

Una ventana se visualiza utilizando una función EGL cuyo nombre empieza por *ConsoleLib.openWindow*. Si no ha visualizado una ventana al presentar otros datos, EGL le proporciona automáticamente una ventana.

Prompt

Una solicitud es una sentencia en línea que obtiene la entrada de usuario. A continuación se muestra una declaración de una solicitud:

```
myPrompt Prompt { message = "Type your ID: "};
```

Una solicitud se visualiza incluyendo la variable en una sentencia **openUI**, que enlaza la solicitud a una variable de tipo String, pero sólo para entrada. Puede configurar la solicitud para que acepte un único carácter o una serie.

ConsoleField

Un `consoleField` es un campo en pantalla que se declara en el contexto de un formulario de consola (como se describe más adelante). El siguiente ejemplo declara un `consoleField` cuyo contenido puede variar durante la ejecución:

```
myField ConsoleField (  
    name="myFieldName",  
    position=[1,31],  
    fieldLen=20,  
    binding = "myVariable" );
```

Para especificar texto constante, utilice un asterisco (*) en lugar del nombre de variable, como en el siguiente ejemplo:

```
* ConsoleField  
{ position=[2,5], value="Title: " };
```

Se recomienda encarecidamente que cuando declare un `consoleField` con nombre, utilice el mismo nombre para el `consoleField` y para el valor del atributo de nombre del `consoleField`. Sin embargo, se permiten nombres distintos para estos dos usos. Podría hacer referencia al nombre de `consoleField` (como por ejemplo, *myField*) cuando el acceso al `consoleField` se resuelve durante la generación. Podría hacer referencia al valor de atributo de nombre (como por ejemplo, *myFieldName*) cuando el acceso se resuelve durante la ejecución, como ocurre cuando el `consoleField` se utiliza para definir un evento en la sentencia **openUI**.

ConsoleForm

Un `consoleForm` es básicamente un conjunto de `consoleFields`. Para activar un `consoleForm`, invoque la función de sistema **ConsoleLib.displayForm**. Para visualizar un `consoleForm` de sólo lectura, por ejemplo, puede realizar lo siguiente:

1. Invoque **ConsoleLib.displayForm**
2. Invoque la función de sistema **ConsoleLib.getKey** para que espere una pulsación de usuario

En cambio, para permitir que el usuario escriba en un `consoleField`, realice lo siguiente:

1. Invoque **ConsoleLib.displayForm**
2. Emita una sentencia **openUI** que haga referencia al `consoleForm` visualizado o a `consoleFields` específicos del `consoleForm`.

El `consoleForm` es un registro de subtipo `ConsoleForm` y, aparte de incluir `consoleFields`, también puede incluir cualquiera de los campos que son válidos en cualquier registro EGL.

Para permitir la interacción de usuario con una tabla en pantalla de `consoleFields`, realice lo siguiente:

1. En el `consoleForm`, declare un `arrayDictionary` que a su vez haga referencia a matrices de `consoleField` que también estén declaradas en el `consoleForm`
2. Utilice dicho `arrayDictionary` en una sentencia **openUI**

Para permitir la interacción de usuario con sólo un subconjunto de `consoleFields` del `consoleForm`, puede listar los `consoleFields` en la sentencia **openUI**, ya sea explícitamente o haciendo referencia a un diccionario. Al igual que el `arrayDictionary`, el diccionario se declara en el `consoleForm` y hace referencia a los `consoleFields` que también están declarados en el `consoleForm`.

EGL no visualiza ninguna variable primitiva que se declara en el `consoleForm`. Puede utilizar dicha variable para enlazar un `consoleField`, ya que se puede utilizar una variable declarada fuera del `consoleForm`.

En general, puede crear enlaces de `consoleForm` de una de las dos maneras siguientes:

- Estableciendo un enlace predeterminado al declarar el `consoleForm`.
- Estableciendo un enlace al codificar la sentencia **openUI**.

Cualquier enlace especificado en la sentencia **openUI** altera temporalmente el enlace predeterminado en su conjunto; no queda ninguno de los enlaces de la declaración de `consoleForm`.

Si utiliza la sentencia **openUI** para enlazar variables, una opción es utilizar la propiedad **isConstruct** de la sentencia, que actúa de la manera siguiente:

- Da formato a la entrada de usuario convirtiéndola en una serie apropiada para una cláusula SQL WHERE
- Coloca dicha serie en una única variable para que pueda codificar fácilmente una sentencia SQL SELECT que recupera los datos solicitados por el usuario de una base de datos relacional, del mismo modo que cuando se codifica una sentencia EGL **prepare**

Para obtener información detallada sobre la propiedad **isConstruct**, consulte el apartado *Sentencia OpenUI*.

El *orden de tabulación* es el orden en que el usuario pasa de un `consoleField` a otro. De forma predeterminada, el orden de tabulación es el orden de los `consoleFields` en la declaración de `consoleForm`. Si incluye una lista de `consoleFields` en una sentencia **openUI**, el orden de tabulación es el orden de los `consoleFields` en dicha sentencia; de forma similar, si incluye un diccionario o `arrayDictionary` en una sentencia **openUI**, el orden de tabulación es el orden de los `consoleFields` en la declaración del diccionario o `arrayDictionary`.

De forma predeterminada, el usuario sale de una sentencia **openUI** relacionada con `consoleForm` pulsando la tecla **Esc**.

Menu

Un menú es un conjunto de etiquetas visualizadas horizontalmente. Una etiqueta es para el menú en su conjunto y una para cada `menuItem` del menú. Para garantizar que se obtiene una respuesta cuando el usuario selecciona un determinado `menuItem`, haga referencia al menú en su conjunto en la sentencia **openUI** y haga referencia al `menuItem` en una cláusula `OnEvent` de dicha sentencia.

MenuItem

Un menuItem muestra una etiqueta y se utiliza tal como se describe en el apartado anterior.

Conceptos relacionados

“ArrayDictionary” en la página 86

“Interfaz de usuario de consola” en la página 177

“Diccionario” en la página 82

Consulta relacionada

“Propiedades y campos de ConsoleField” en la página 441

“Propiedades de ConsoleForm en consoleUI de EGL” en la página 454

“Biblioteca ConsoleLib de EGL” en la página 757

“Opciones de pantalla ConsoleUI para UNIX” en la página 184

“Campos de Menu en consoleUI de EGL” en la página 455

“Campos de MenuItem en consoleUI de EGL” en la página 456

“openUI” en la página 620

“Campos de PresentationAttributes en consoleUI de EGL” en la página 458

“Campos de Prompt en consoleUI de EGL” en la página 460

“Campos de Window en consoleUI de EGL” en la página 461

Tareas relacionadas

“Crear una interfaz con consoleUI” en la página 178

Utilización de new en ConsoleUI

Cuando se crea un programa EGL que utiliza consoleUI, cada variable de tipo Menu, MenuItem, Prompt y Window es una *variable de referencia*, que contiene una dirección de memoria que hace referencia a un valor almacenado fuera de la variable.

Puede declarar una variable de referencia de la misma manera que declara cualquier otra variable, como se muestra en el siguiente ejemplo:

```
myPrompt Prompt { message = "Type your ID: "};
```

Como alternativa, puede declarar una variable de referencia e inicializarla con la palabra reservada **new**, como se muestra en el siguiente ejemplo:

```
myPrompt Prompt = new Prompt { message = "Type your ID: "};
```

Cuando se declaran variables, la diferencia entre los dos formatos tiene poco efecto práctico; sin embargo, cuando se codifica la sentencia openUI, la palabra **new** proporciona comodidad en la codificación, como se muestra en *openUI*.

La sintaxis general de **new** es la siguiente:

```
new nombreComponente
```

nombreComponente

Una de las siguientes palabras, que hacen referencia a una determinada clase de componente:

- Menu
- MenuItem
- Prompt
- Window

Para obtener información detallada sobre otras implicaciones de las variables de referencia, consulte el apartado *Compatibilidad de referencia en EGL*.

Conceptos relacionados

“Interfaz de usuario de consola” en la página 177

Consulta relacionada

“Componentes de ConsoleUI y variables relacionadas” en la página 180

“openUI” en la página 620

“Compatibilidad de referencia en EGL” en la página 739

Tareas relacionadas

“Crear una interfaz con consoleUI” en la página 178

Opciones de pantalla ConsoleUI para UNIX

Lo usuario de EGL en las plataformas UNIX soportadas tienen capacidad de ejecutar la aplicación ConsoleUI utilizando una modalidad gráfica o una modalidad curses de UNIX.

Modalidad de visualización gráfica

Para ejecutar una aplicación ConsoleUI en modalidad de visualización gráfica debe asegurarse de que la biblioteca curses de EGL no esté ubicada en la variable de entorno Library Path del shell que se está ejecutando. Esta es la modalidad predeterminada.

Modalidad curses de UNIX

Para ejecutar una aplicación ConsoleUI en la modalidad curses de UNIX, debe tener la biblioteca curses de EGL específica de plataforma adecuada en la variable de entorno Library Path del shell que se está ejecutando. Las bibliotecas curses de EGL deben bajarse del sitio Web de soporte de EGL.

Para bajar la biblioteca curses de EGL:

1. Vaya al sitio Web de soporte de EGL adecuado.
 - El URL de Rational Application Developer es:
`http://www3.software.ibm.com/ibmdl/pub/software/rational/sdp/rad/60/redist`
 - El URL de Rational Web Developer es:
`http://www3.software.ibm.com/ibmdl/pub/software/rational/sdp/rwd/60/redist`
 - .
2. Descargue el archivo **EGLRuntimesV60IFix001.zip** en el directorio que prefiera.
3. Desempaque el archivo **EGLRuntimesV60IFix001.zip** para identificar los archivos siguientes:
 - AIX: **EGLRuntimes/Aix/bin/libCursesCanvas6.so**
 - Linux: **EGLRuntimes/Linux/bin/libCursesCanvas6.so**
4. Inserte la biblioteca curses de EGL adecuada en la variable de entorno Library Path.

AIX: Establezca la variable de entorno Library Path **'LIBPATH'** utilizando el bourne-shell siguiente:

```
"LIBPATH=$INSDIR/aix; export LIBPATH"
```

Linux: Establezca la variable de entorno Library Path 'LD_LIBRARY_PATH' utilizando el bourne-shell siguiente:

```
"LD_LIBRARY_PATH=$INSTRDIR/aix; export LD_LIBRARY_PATH"
```

Conceptos relacionados

"Interfaz de usuario de consola" en la página 177

Consulta relacionada

"Biblioteca ConsoleLib de EGL" en la página 757

"Componentes de ConsoleUI y variables relacionadas" en la página 180

"openUI" en la página 620

Tareas relacionadas

"Crear una interfaz con consoleUI" en la página 178

Crear una aplicación Web de EGL

Soporte Web

EGL proporciona el siguiente soporte para aplicaciones basadas en Web:

- Puede desarrollar un *PageHandler*, que es un componente lógico en que cada función del mismo se invoca a través de una determinada acción de usuario en una página Web. La generación de este componente lógico también puede proporcionar un JSP de JavaServer Faces para la personalización.
- Puede proporcionar una funcionalidad que sea común a varias páginas Web, como por ejemplo cuando cada página de una aplicación muestra un botón que el usuario pulsa para finalizar la sesión. En este caso, la pulsación del usuario puede invocar un programa EGL, que actúa como una subrutina común.
- Finalmente, cuando trabaje en WebSphere Page Designer, puede personalizar los JSP de JavaServer Faces y puede afectar a los *PageHandlers*, como se describe en la sección *Soporte de Page Designer para EGL*.

Conceptos relacionados

“PageHandler” en la página 194

“WebSphere Application Server y EGL” en la página 341

Tareas relacionadas

“Iniciar una aplicación Web en el sistema local” en la página 340

Consulta relacionada

“Soporte de Page Designer para EGL” en la página 192

Crear una aplicación Web EGL de tabla única

Asistente Páginas y componentes de datos de EGL

El asistente Páginas y componentes de datos de EGL proporciona una forma práctica de crear un programa de utilidad basado en Web que permita mantener una tabla específica en una base de datos relacional.

El asistente crea estas entidades:

- Un conjunto de *PageHandlers* que genera posteriormente en un conjunto de componentes que se ejecutan en Java Server Faces
- Un componente de registro SQL, así como los componentes de elemento de datos relacionados y los componentes de función basados en biblioteca
- Un conjunto de archivos JSP que proporciona las páginas Web siguientes:
 - Una *página de condición de selección* que acepta los criterios de selección del usuario
 - Una *página de lista* que muestra varias filas, basadas en los criterios del usuario
 - Una *página de creación de detalles* que permite al usuario visualizar o insertar una fila

- Una *página de detalles* que permite al usuario visualizar, actualizar o suprimir una fila

El usuario encuentra primero la página de criterios de selección pero si no puede especificar la información necesaria para esa página, el usuario encuentra primero la página de lista, que proporciona acceso (en esta situación) a cada fila de la tabla.

Al trabajar con el asistente, puede hacer lo siguiente:

- Personalice las páginas Web descritas anteriormente, activando o desactivando los campos visualizados o incluir enlaces de una página a otra.
- Especifique los campos de clave de registro SQL que se utilizan para crear, leer, actualizar o suprimir una fila de una tabla o vista de base de datos dada.
- Personalice las sentencias SQL explícitas para crear, leer o actualizar una fila. (La sentencia SQL para suprimir una fila no puede personalizarse.)
- Especifique los campos de clave de registro SQL que se utilizan para seleccionar un conjunto de filas de una base de datos o vista dada.
- Personalice una sentencia SQL explícita para seleccionar un conjunto de filas.
- Valide y ejecute cada sentencia SQL

La salida incluye estos archivos:

- Un archivo HTML (index.html) que invoca la aplicación Web.
- Un conjunto de archivos JSP que proporciona las páginas Web descritas anteriormente.
- Un archivo fuente EGL que contiene todos los componentes de elemento de datos a los que hacen referencia los elementos de estructura en los componentes de registro SQL.
- Para cada componente de registro SQL, el asistente genera también dos archivos: uno para el mismo componente de registro y otro para las funciones relacionadas basadas en biblioteca. Puede reducir el número de archivos si marca el recuadro de selección **Registro y biblioteca en el mismo archivo**.

Puede personalizar el programa de utilidad basado en Web una vez lo cree el asistente.

Conceptos relacionados

“Programa Java, PageHandler y biblioteca” en la página 328

“Soporte de SQL” en la página 229

Tareas relacionadas

“Crear una aplicación Web EGL de tabla única”

“Crear, editar o suprimir conexión base datos para asistentes EGL” en la página 257

“Personalizar sentencias SQL en los asistentes de EGL” en la página 257

“Definir páginas Web en el asistente Páginas y componentes de datos EGL” en la página 190

Crear una aplicación Web EGL de tabla única

Para crear una aplicación Web EGL a partir de una sola tabla de base de datos relacional, haga lo siguiente:

1. Seleccione **Archivo > Nuevo >Otros....** Se visualiza un diálogo para seleccionar un asistente.
2. Expanda **EGL** y efectúe una doble pulsación sobre **Páginas y componentes de datos EGL**. Se visualiza el diálogo Páginas y componentes de datos EGL.

3. Especifique un nombre de proyecto Web EGL o seleccione un proyecto existente de la lista desplegable. Los componentes EGL se generarán en este proyecto.
4. Seleccione una conexión de base de datos existente de la lista desplegable o establezca una nueva:
 - Para establecer una conexión de base de datos nueva, pulse **Añadir** y siga las instrucciones proporcionadas en el tema de ayuda *Página de conexión de base de datos* a la que puede acceder pulsando F1
 - Para obtener detalles sobre la edición o supresión de una conexión de base de datos, consulte la sección *Crear, editar o suprimir una conexión de base de datos para los asistentes de EGL*

Cuando se establece una conexión con la base de datos, se visualiza una lista de tablas de base de datos.
5. Sino desea aceptar el nombre del archivo EGL por omisión para elementos de datos, teclee un nombre de archivo nuevo.
6. En el campo **Seleccione los datos**, pulse el nombre de la tabla de base de datos deseada.
7. En el campo **Nombre de registro**, especifique el nombre del registro EGL a crear o acepte el nombre por omisión.
8. Si desea incluir el componente de biblioteca y los componentes de registro de SQL en el mismo archivo, marque el recuadro de selección.
9. Para establecer campos adicionales en valores no predeterminados, pulse **Siguiente**; de lo contrario, pulse **Finalizar**. En los pasos restantes se da por supuesto que ha pulsado **Siguiente**.
10. Seleccione el campo de clave que se utilizará al leer, actualizar y suprimir filas individuales y pulse la flecha derecha. Para seleccionar varios campos de clave, mantenga pulsada la tecla **Ctrl** mientras pulsa sobre distintos nombres de campo. Para eliminar un campo de clave de la lista de la derecha, resalte el nombre del campo y pulse la flecha izquierda.
11. Elija el campo de condición de selección que se utilizará al seleccionar un conjunto de filas y pulse la flecha derecha. Para seleccionar varios campos, mantenga pulsada la tecla **Ctrl** mientras pulsa sobre distintos nombres de campo. Para eliminar un campo de la lista de la derecha, resalte el nombre del campo y pulse la flecha izquierda.
12. Para personalizar las sentencias SQL implícitas, consulte la sección *Personalizar sentencias SQL en los asistentes EGL*. Esta opción no está disponible para la sentencia **delete** de EGL.
13. Pulse en **Siguiente**.
14. Si desea aplicar una plantilla a las páginas Web nuevas, siga estos pasos:
 - a. Marque el recuadro de selección **Seleccionar plantilla de página**.
 - b. Seleccione un tipo de plantilla de página pulsando **Plantilla de página de ejemplo** o **Plantilla de página definida por el usuario**.
 - c. Pulse sobre la plantilla de página que desee utilizar. Puede seleccionar una miniatura o buscar la plantilla pulsando el botón **Examinar**.
15. Pulse en **Siguiente**.
16. Para personalizar las páginas Web, consulte la sección *Definir páginas Web en el asistente Páginas y componentes de datos EGL*.
17. Pulse en **Siguiente**.
18. Se visualiza la pantalla Generar la aplicación Web, incluyendo (en la parte inferior) una lista de los archivos y páginas Web que se generarán:

- a. Para cambiar el nombre del proyecto Web EGL que recibirá los componentes EGL, teclee un nombre de proyecto en el campo **Nombre de proyecto Web EGL** o seleccione un proyecto de la lista desplegable relacionada.
- b. Para especificar los paquetes EGL y Java para un tipo específico de componente (PageHandler, datos o biblioteca), teclee un nombre de paquete en el campo relacionado o seleccione un nombre de la lista desplegable relacionada.
- c. Para cambiar el nombre de los archivos JSP y EGL generados para una página Web dada, pulse la entrada adecuada bajo **Páginas Web** y teclee el nombre nuevo. Cada nombre de archivo incluye las letras o números especificados, pero excluye los espacios y otros caracteres.

Teclee o seleccione paquetes para los componentes de PageHandler, componentes de datos y componentes de biblioteca.

19. Pulse en **Finalizar**.

Conceptos relacionados

“Soporte de SQL” en la página 229

Tareas relacionadas

“Crear, editar o suprimir conexión base datos para asistentes EGL” en la página 257

“Crear componentes datos EGL de tablas de bases datos relacionales” en la página 255

“Personalizar sentencias SQL en los asistentes de EGL” en la página 257

“Definir páginas Web en el asistente Páginas y componentes de datos EGL”

Definir páginas Web en el asistente Páginas y componentes de datos EGL

El asistente Páginas y componentes de datos EGL crea una aplicación Web a partir de una tabla de base de datos relacional. En este asistente puede especificar los aspectos siguientes de cada tipo de página Web que se genera:

- Título de página
- Hoja de estilos
- Campos a visualizar, incluyendo el orden y las propiedades
- Enlaces a otras páginas

En el diálogo del asistente llamado *Definir las páginas Web de la aplicación*, puede pulsar las pestañas para navegar entre páginas. Haga lo siguiente para cada página (cuando sea posible):

1. Establezca el título de la página en el campo **Título de página**
2. Seleccione una hoja de estilos de una lista desplegable en el campo **Hoja de estilos**
3. Para ver el efecto de aceptar la definición de página actual, pulse **Vista previa**.
4. Si desea especificar el número de filas que se van a visualizar en una página, seleccione **Paginación** y asigne un número de filas (un entero positivo) a **Tamaño de página**.
5. Seleccione los campos que deben incluirse en la página:
 - a. Para incluir un campo, marque el recuadro de selección relacionado. Para seleccionar cada campo, pulse **Todos**.
 - b. Para excluir un campo, quite la marca del recuadro de selección relacionado. Para excluir cada campo, pulse **Ninguno**.

- c. Para cambiar la ubicación de visualización de un campo, pulse sobre el campo y utiliza las flechas Arriba y Abajo para mover el campo a otra ubicación.
 - d. Para establecer las propiedades de un campo, pulse sobre el campo y efectúe una doble pulsación sobre el campo **Valor** en el panel Propiedades. Especifique el valor o, en algunos casos, seleccione de una lista desplegable.
6. Seleccione las acciones que el usuario puede realizar en la página:
- a. Para incluir una acción, marque el recuadro de selección relacionado. Seleccionar cada acción, pulse **Seleccionar todo**.
Las acciones individuales son **crear, suprimir, extraer, listar y leer**:
 - **Crear** enlaza con la página de creación de detalles en la que el usuario puede visualizar o insertar una fila. La opción está presente en la página de detalles solo para indicar que el usuario puede crear un fila a partir de esa página.
 - **Suprimir** solo está disponible en la página de detalles. Esta opción suprime el registro que tiene la clave especificada por el usuario.
 - **Extraer** enlaza con la página de condición de selección que acepta criterios de selección del usuario. La opción está presente en la página de condición de selección solo para indicar que el usuario puede originar la devolución de un conjunto de resultados de esa página.
 - **Listar** enlaza con la página de lista que visualiza varias filas de acuerdo con los criterios del usuario.
 - **Leer** solo está disponible en la página de creación de detalles. Esta opción visualiza el registro que tiene la clave especificada por el usuario.
 - **Actualizar** solo está disponible en la página de detalles. Esta opción actualiza el registro modificado por el usuario.
 - b. Para excluir una acción, quite la marca del recuadro de selección relacionado. Para excluir cada acción, pulse **Ninguna**.
Si una acción dada está siempre disponible, no podrá quitar la marca del recuadro de selección.
 - c. Para establecer la etiqueta de página Web para una acción, pulse sobre el nombre de acción y efectúe una doble pulsación sobre el campo **Valor** en el panel Propiedades y especifique un valor.

Conceptos relacionados

“Soporte de SQL” en la página 229

“Asistente Páginas y componentes de datos de EGL” en la página 187

Tareas relacionadas

“Crear una aplicación Web EGL de tabla única” en la página 188

“Crear componentes datos EGL de tablas de bases datos relacionales” en la página 255

“Crear, editar o suprimir conexión base datos para asistentes EGL” en la página 257

“Establecer preferencias de EGL” en la página 113

“Iniciar una aplicación Web en el sistema local” en la página 340

Crear un componente EGL pageHandler

Un componente pageHandler controla la interacción de tiempo de ejecución de un usuario con una página Web proporcionando datos y servicios a un JSP de Java Server Faces. Cuando crea un JSP, se crea automáticamente un componente pageHandler en un paquete llamado *pagehandlers* dentro de la carpeta *EGLSource*. El nombre del componente pageHandler es el mismo que el del JSP correspondiente, pero con la extensión de archivo .egl.

También puede crear un componente `pageHandler` y dejar que el sistema añada automáticamente el JSP al proyecto, suponiendo que no exista ya un archivo JSP con el mismo nombre en el proyecto Web de EGL. Para crear un componente `pageHandler` de EGL, haga lo siguiente:

1. Si el proyecto Web de EGL no contiene un paquete llamado *pagehandlers*, debe crear uno. Page Designer necesita que todos los componentes de `pageHandler` residan en un paquete llamado *pagehandlers*. Para obtener detalles sobre la creación de paquetes, consulte la sección *Crear un paquete de EGL*.
2. Busque un archivo de EGL en el paquete *pagehandlers* para que albergue el componente `pageHandler`. Abra el archivo en el editor de EGL. Debe crear un archivo EGL si todavía no tiene uno.
3. Teclee las especificaciones del componente `pageHandler` de acuerdo con la sintaxis de EGL (para conocer los detalles, consulte la sección *Componente PageHandler en formato de código fuente EGL*). Puede utilizar la ayuda de contenido para incluir un esquema de la sintaxis del componente `pageHandler` en el archivo.
4. Guarde el archivo EGL.

Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13

“PageHandler” en la página 194

Tareas relacionadas

“Crear un paquete de EGL” en la página 128

“Crear un archivo fuente EGL” en la página 128

“Utilizar las plantillas EGL con la ayuda de contenido” en la página 129

“Utilizar la vista Edición rápida para el código de PageHandler” en la página 202

Consulta relacionada

“Ayuda de contenido en EGL” en la página 483

“Convenios de denominación” en la página 672

“Componente PageHandler en formato fuente EGL” en la página 679

Soporte de Page Designer para EGL

Cuando crea un archivo JSP en un proyecto Web de EGL, EGL crea automáticamente un `PageHandler` y ese `PageHandler` incluye código EGL de esqueleto para que usted lo personalice. En Page Designer, haga lo siguiente:

1. Arrastre componentes de la paleta a un JSP
2. Utilice la vista Atributos para establecer características específicas de componente, como por ejemplo color y para establecer *enlaces*, que son relaciones entre componentes y datos o lógica

Puede realizar tareas específicas de EGL:

- Crear variables de EGL y situarlas en un `PageHandler` existente.
- Enlazar elementos de `PageHandler` con componentes de interfaz de usuario JSP.
- Enlazar funciones de `PageHandler` con botones y controles de hiperenlace. Las funciones actúan como manejadores de eventos.

Al utilizar la pestaña de código fuente en Page Designer, puede enlazar manualmente componentes de un archivo JSP (específicamente de un archivo JavaServer Faces) con áreas de datos y funciones de un `PageHandler`. Aunque EGL no es sensible a las mayúsculas y minúsculas, los nombres de EGL a los que se hace referencia en el archivo JSP deben coincidir en cuanto a mayúsculas y

minúsculas con la declaración de variable o función EGL; si la coincidencia no es total, se produce un error de JavaServer Faces. Es recomendable no cambiar las mayúsculas y minúsculas de una variable o función EGL después de enlazar esa variable o función con un campo JSP.

Para conocer más detalles sobre la denominación, consulte el apartado *Cambios en identificadores de EGL en archivos JSP y beans Java generados*.

Enlazar los componentes con las áreas de datos del PageHandler

La mayoría de los componentes del JSP mantienen una correspondencia de uno a uno con los datos. Un recuadro de texto, por ejemplo, muestra el contenido del elemento EGL al que está enlazado. Un recuadro de texto de entrada también actualiza el elemento EGL si el usuario cambia los datos.

Una situación más compleja tiene lugar cuando el usuario especifica un grupo de recuadros de selección, un recuadro de lista, un grupo de botones de selección o un recuadro combinado. En tales casos, son necesarios dos tipos de enlaces diferentes:

- Uno destinado a enlazar el componente con el texto que el usuario debe visualizar. Un ejemplo de ello es el texto de un elemento de un recuadro de lista.
- Otro destinado a enlazar el componente con un área de datos del PageHandler que recibe un valor para indicar la elección del usuario. Puede crear un elemento de datos, por ejemplo, que debe recibir el índice numérico de un elemento de recuadro de lista seleccionado por el usuario.

En la vista Propiedades, puede utilizar dos procedimientos para enlazar el componente con el texto que el usuario visualiza:

- Puede utilizar **Añadir opción** para indicar que el componente está asociado con una sola serie de caracteres, que puede especificarse explícitamente o identificarse mediante un elemento del PageHandler
- Puede utilizar **Añadir conjunto de opciones** para indicar que el componente está asociado con una lista de series de caracteres, que puede especificarse explícitamente o identificarse mediante un área del PageHandler, como por ejemplo una tabla de datos o una matriz de elementos de carácter

Como alternativa, puede enlazar un componente de selección única (recuadro combinado, recuadro de lista de selección única o grupo de botones de selección) con una matriz de elementos de carácter arrastrando la matriz desde la vista Datos de página hasta el componente.

Para enlazar un componente con un área de datos que va a recibir un valor que indicará la elección del usuario, puede utilizar la vista Datos de página o la vista Propiedades. El procedimiento es el mismo que para enlazar cualquier componente, incluso un recuadro de texto simple.

Si el valor sólo puede ser una de dos alternativas, puede enlazar el componente con un elemento EGL para el que la propiedad de elemento **boolean** esté establecida en *yes*. El componente rellena el elemento con uno de estos dos valores:

- Para un elemento de caracteres, el valor es **Y** (para sí) o **N** (para no)
- Para un elemento numérico, el valor es **1** (para sí) o **0** (para no)

Cuando se visualiza un recuadro de selección, el estado (seleccionado o no seleccionado) depende del valor del elemento enlazado.

Para obtener detalles acerca de las propiedades que pueden aplicarse a los elementos de datos del PageHandler, consulte el apartado *Propiedades de elemento de página*.

Enlazar los componentes con las funciones

Después de enlazar un botón de mandato o un hiperenlace de mandato a la superficie de la página, puede enlazar ese componente con una función EGL existente o con un manejador de eventos creado por Page Designer:

- Puede enlazar el componente con un manejador de eventos existente de cualquiera de estas formas:
 - Arrastrando la función EGL del nodo Acciones de la vista Datos de página hasta el componente, tal como se recomienda
 - Abriendo el componente en la vista Edición rápida
 - Pulsando el componente con el botón derecho del ratón y seleccionando **Editar evento de mandato Faces**
- Puede hacer que Page Designer cree un manejador de eventos nuevo cuando el usuario abra el componente en la vista Edición rápida o cuando pulsa el componente con el botón derecho del ratón y selecciona **Editar evento de mandato Faces**

Si Page Designer crea un manejador de eventos en el PageHandler y le proporciona acceso a esa función del PageHandler, el nombre de la función es el ID de botón asignado por la herramienta más la serie "Action". Si el nombre no es exclusivo del PageHandler, Page Designer añade un número al final del nombre de función.

Conceptos relacionados

"PageHandler"

Tareas relacionadas

"Crear un campo de EGL y asociarlo con un JSP Faces" en la página 199

"Asociar un registro EGL con un JSP Faces" en la página 200

"Utilizar la vista Edición rápida para el código de PageHandler" en la página 202

Consulta relacionada

"Componente PageHandler en formato fuente EGL" en la página 679

"Propiedades del campo PageHandler" en la página 685

PageHandler

Un *PageHandler* de EGL es un ejemplo de *código de página*; controla la interacción de tiempo de ejecución de un usuario con una página Web y puede realizar cualquiera de las siguientes tareas:

- Asignar valores de datos para someterlos a un archivo JSP. Estos valores se visualizan finalmente en una página Web.
- Cambiar los datos devueltos del usuario o de un programa llamado.
- Pasar el control a otro JSP.

Puede trabajar más fácilmente personalizando un archivo JSP y creando el PageHandler en Page Designer; para conocer detalles, consulte *Soporte de Page Designer para EGL*.

El propio PageHandler incluye variables y los siguientes tipos de lógica:

- Una función `OnPageLoad`, que se invoca la primera vez que el JSP muestra la página Web
- Un conjunto de funciones de manejador de eventos, cada una de las cuales se invoca como respuesta a una determinada acción del usuario (concretamente, cuando el usuario pulsa un botón o un enlace de hipertexto)
- Opcionalmente, las funciones de validación que se utilizan para validar los campos de entrada de la página Web
- Las funciones privadas que sólo pueden invocarse mediante otras funciones del `PageHandler`

Existen dos maneras de acceder a las variables del `PageHandler`:

- El entorno de ejecución accede a los datos automáticamente. Si un campo del JSP está *enlazado* a un elemento del `PageHandler`, el resultado es el siguiente:
 - Una vez que se ha ejecutado la función `OnPageLoad` y antes de que se visualice la página Web, cada valor de elemento de `PageHandler` se escribe en el campo del JSP en el que están enlazados los datos.
 - Cuando el usuario envía un formulario en el que residen los campos enlazados del JSP, el valor de cada campo del formulario enviado se copia en el elemento del `PageHandler` asociado. Sólo entonces el control se pasa a un manejador de eventos. (Sin embargo, esta descripción no incluye los pasos de validación, que se tratarán más adelante en este tema).
- Los manejadores de eventos y la función `OnPageLoad` también pueden interactuar con los datos, así como con los almacenes de datos (como por ejemplo, las bases de datos SQL) y con los programas llamados.

El componente `pageHandler` debe ser simple. Aunque el componente podría incluir validaciones de datos ligeros, como por ejemplo comprobaciones de rangos, se aconseja que invoque otros programas para ejecutar lógica empresarial compleja. El acceso a bases de datos, por ejemplo, debe reservarse a un programa llamado.

Salida asociada a un `PageHandler`

Cuando guarda un `PageHandler`, EGL coloca un archivo JSP en la carpeta de proyecto `WebContent\WEB-INF`, pero sólo en este caso:

- Ha asignado un valor a la propiedad **view** del `PageHandler`, que especifica un nombre de archivo JSP
- La carpeta `WebContent\WEB-INF` no contiene un archivo JSP con el nombre especificado

EGL no sobrescribe nunca un archivo JSP.

Si una preferencia del entorno de trabajo está establecida en construcción automática al guardar, se produce la generación de `PageHandler` siempre que se guarda el `PageHandler`. En cualquier caso, cuando se genera un `PageHandler`, la salida se compone de los siguientes objetos:

- El *bean de página* es una clase Java que contiene datos y que proporciona servicios de inicialización, validación de datos y manejo de eventos para la página Web.
- Un elemento de `<bean gestionado>` se coloca en el archivo de configuración JSF del proyecto para identificar el bean de página durante la ejecución.
- Un elemento de `<norma de navegación>` se crea en el archivo de configuración de la aplicación JSF para asociar un resultado JSF (el nombre del `PageHandler`) con el archivo JSP que debe invocarse.
- Un archivo JSP, en la misma situación que cuando guarda el `PageHandler`.

También se generan todas las tablas de datos y registros que utiliza el manejador de componentes.

Validación

Si los códigos JSF basados en JSP realizan conversión de datos, validación de datos o manejo de eventos, el entorno de ejecución JSF realiza el proceso necesario en cuanto el usuario envía la página Web. Si se encuentran errores, el entorno de ejecución JSF puede volver a visualizar la página sin pasar el control al PageHandler. Sin embargo, si recibe el control, el PageHandler puede llevar a cabo un conjunto de validaciones basadas en EGL.

Las validaciones basadas en EGL se realizan si se especifican los siguientes detalles al declarar el PageHandler:

- Las ediciones básicas (como, por ejemplo, la longitud mínima de entrada) para campos de entrada individuales.
- Las ediciones basadas en el tipo (carácter, numérico) para campos individuales.
- Las ediciones de DataTable (rango, coincidencia válida y coincidencia no válida) para campos de entrada individuales, como se explica en la sección *Componente DataTable*.
- Las funciones de edición para campos de entrada individuales.
- La función de edición para el PageHandler en su conjunto.

El PageHandler supervisa las ediciones en el siguiente orden, pero sólo para los elementos cuyos valores ha cambiado el usuario:

1. Todas las ediciones básicas y basadas en el tipo, aunque algunas no sean satisfactorias
2. (Si las ediciones anteriores han sido satisfactorias) todas las ediciones de tabla, aunque algunas no sean satisfactorias
3. (Si las ediciones anteriores han sido satisfactorias) todas las funciones de edición de tabla, aunque algunas no sean satisfactorias
4. (Si todas las ediciones anteriores han sido satisfactorias) la función de edición de pageHandler

La propiedad del elemento de página **validationOrder** define el orden en el que se editan los campos de entrada individuales y en el que se invocan las funciones de validador de campos.

Si no se especifican propiedades **validationOrder**, el valor predeterminado es el orden de los elementos definidos en el PageHandler, de arriba a abajo. Si **validationOrder** se ha definido para algunos de los elementos de un PageHandler, pero no para todos, primero se realiza la validación de todos los elementos con la propiedad **validationOrder**, en el orden especificado. A continuación, se realiza la validación de los elementos sin la propiedad **validationOrder** en el orden de los elementos en el PageHandler, de arriba a abajo.

Caso práctico de entorno de ejecución

Esta sección ofrece una visión general técnica de la interacción en tiempo de ejecución entre el usuario y el servidor de aplicaciones Web.

Cuando el usuario invoca un JSP que está soportado por un PageHandler, se realizan los pasos siguientes:

1. El servidor de aplicaciones Web inicializa el entorno:

- a. Construye un objeto de sesión para conservar los datos que necesitan las aplicaciones a las que accede el usuario en varias interacciones
 - b. Construye un objeto de petición para conservar los datos sobre la interacción actual del usuario
 - c. Invoca el JSP
2. El JSP procesa una serie de códigos JSF para construir una página Web:
 - a. Crea una instancia del PageHandler, hace que la función onPageLoad (si existe) se invoque con argumentos especificados por el usuario y coloca el PageHandler en el objeto de petición
 - b. Accede a los datos almacenados en los objetos de petición y sesión para incluirlos en la página Web

Nota: El componente pageHandler tiene una propiedad llamada onPageLoadFunction, que identifica la función de PageHandler que se invoca cuando se inicia el JSP. La función recupera automáticamente los argumentos suministrados por el usuario que se han pasado a la misma; puede llamar a otro código; y puede colocar datos adicionales en el objeto de petición o sesión del servidor de aplicaciones Web; pero la función no puede reenviar el control a otra página ni hacer que se visualice un mensaje de error cuando la página se presenta por primera vez al usuario.

3. El JSP envía la página Web al usuario y el servidor de aplicaciones Web destruye el objeto de respuesta, dejando el objeto de sesión y el JSP.

Si el usuario suministra datos en los campos que se visualizan en pantalla asociados a un código HTML <FORM> y envía el formulario, se realizan los pasos siguientes:

1. El servidor de aplicaciones Web reinicializa el entorno:
 - a. Construye un objeto de petición
 - b. Coloca los datos recibidos para el formulario enviado en el bean de página para validarlos
 - c. Vuelve a invocar el JSP
2. El JSP procesa una serie de códigos JSF para almacenar los datos recibidos en el bean de página.
3. El PageHandler de tiempo de ejecución valida los datos:
 - a. Realiza ediciones relativamente básicas (como, por ejemplo, la longitud mínima de entrada), según se especifica en las declaraciones de datos de pageHandler
 - b. Invoca funciones de validación específicas del elemento, según se especifica en las declaraciones de datos de pageHandler
 - c. Invoca una función de validador de pageHandler, ya que es necesaria si desea validar un campo al menos parcialmente en función del contenido de otro campo

(Para obtener información detallada sobre la validación, consulte la sección anterior).

4. Si se produce un error, el entorno de ejecución EGL coloca los errores en una cola JSF y el JSP vuelve a visualizar la página Web con los mensajes incorporados. Sin embargo, si no se produce un error, el resultado es el siguiente:
 - a. Los datos almacenados en el bean de página se escriben en el bean de registro

- b. El proceso subsiguiente está determinado por un manejador de eventos, que se identifica en el código JSF que está asociado al botón que pulsa el usuario o al hipervínculo.

El manejador de eventos puede reenviar el proceso a una etiqueta JSF, que identifica una correlación en un archivo de configuración basado en JSF de tiempo de ejecución. A su vez, la correlación identifica el objeto que debe invocarse, que puede ser un JSP (normalmente un JSP asociado a un PageHandler EGL) o un servlet.

Conceptos relacionados

“Referencias a componentes” en la página 21

“Soporte Web” en la página 187

Consulta relacionada

“Soporte de Page Designer para EGL” en la página 192

“Componente PageHandler en formato fuente EGL” en la página 679

“Propiedades del campo PageHandler” en la página 685

Controles de JavaServer Faces y EGL

JavaServer Faces (JSF) es una infraestructura de componentes de interfaz de usuario del lado del servidor. En términos sencillos, JSF es un conjunto de herramientas y componentes que permiten crear interfaces para páginas Web. Los componentes JSF pueden visualizar datos en una página Web y aceptar una entrada del usuario.

En este tema se explica la relación entre componentes de JSF y EGL. Para obtener más detalles sobre JSF, consulte la sección [Crear aplicaciones Faces - Visión general](#). Para ver una guía de aprendizaje relacionada, pulse **Ayuda > Galería de guías de aprendizaje**, expanda **Aprender practicando** y seleccione *Visualizar información dinámica sobre páginas Web con JavaServer Faces*.

Hay dos métodos disponibles para visualizar datos de EGL en una página Web utilizando controles de JSF:

- Puede crear automáticamente controles de JSF a partir de elementos de datos en la vista Datos de página o a partir de elementos de datos creados en la vista Page Designer. Para utilizar este método, marque el recuadro de selección llamado **Añadir controles para visualizar el elemento EGL en la página Web** cuando siga las instrucciones de las secciones *Asociar un registro EGL con un JSP Faces* o *Crear un elemento de datos EGL y asociarlo con un JSP Faces*.
- Puede añadir controles JSF manualmente y enlazarlos con datos en la vista Datos de página. Este método permite personalizar el diseño de los controles de JSF en la página, en lugar de utilizar el diseño por omisión. Para utilizar este método, vea uno de los temas siguientes:
 - *Enlazar un componente de entrada o salida JavaServer Faces con un PageHandler de EGL*
 - *Enlazar un componente de recuadro de selección de JavaServer Faces con un PageHandler de EGL*
 - *Enlazar un componente de selección única de JavaServer Faces con un PageHandler de EGL*
 - *Enlazar un componente de selección múltiple de JavaServer Faces con un PageHandler de EGL*

También puede enlazar funciones de EGL en PageHandlers con controles de JSF. Consulte la sección *Enlazar un componente de mandato de JavaServer Faces con un PageHandler de EGL*.

Tareas relacionadas

“Crear un campo de EGL y asociarlo con un JSP Faces”

“Asociar un registro EGL con un JSP Faces” en la página 200

“Enlazar componente mandato JavaServer Faces con PageHandler EGL” en la página 201

“Enlazar componente entrada/salida JavaServer Faces con PageHandler EGL” en la página 203

“Enlazar comp. recuadro selección JavaServer Faces con PageHandler EGL” en la página 203

“Enlazar componente selección única JavaServer Faces con PageHandler EGL” en la página 204

“Enlazar comp. selección múltiple JavaServer Faces con PageHandler EGL” en la página 206

Consulta relacionada

“Soporte de Page Designer para EGL” en la página 192

Crear un campo de EGL y asociarlo con un JSP Faces

Para crear un campo primitivo de EGL y asociarlo con un JSP Faces, haga lo siguiente:

1. Abra un archivo JSP Faces en Page Designer. Para abrir un archivo JSP, efectúe una doble pulsación sobre el archivo JSP en el Explorador de proyectos. El JSP se abre en Page Designer. Pulse en la pestaña **Diseño** para acceder a la vista Diseño.

Nota: Puede acceder al PageHandler relacionado pulsando con el botón derecho del ratón en la vista Diseño (o en la vista Fuente) y pulsando **Editar código de página**.

2. En el menú **Ventana**, seleccione **Mostrar vista > Otras > Básica > Paleta**.
3. En la vista Paleta, pulse en **EGL** para visualizar los tipos de objetos de datos de EGL.
4. Arrastre **Campo nuevo** desde la paleta al JSP. Si visualiza un diálogo Campo de datos EGL nuevo.
5. Teclee un nombre de campo en el campo **Nombre**.
6. Seleccione el tipo de campo en la lista desplegable **Tipo** y, si necesita especificar las características primitivas del campo (longitud y posiblemente decimales), teclee la información en el recuadro de texto **Dimensiones**. Se utilizan máscaras predeterminadas si declara elementos de los tipos siguientes:
 - Fecha (máscara *aaaammdd*)
 - Hora (máscara *hhmmss*)
 - Indicación de la hora (máscara *aaaammddhhmmss*)

Si desea especificar un componente DataItem como tipo, seleccione **DataItem**, que es el último valor de la lista. En este caso, se visualiza el diálogo Seleccionar un componente DataItem y puede seleccionar un componente DataItem de la lista o teclear el nombre; a continuación, pulse **Aceptar**.

7. Si está creando una matriz de elementos de datos, marque el recuadro de selección **Matriz** y teclee un entero en el recuadro de selección **Tamaño**.
8. Si no desea incluir el campo en la página, quite la marca del recuadro de selección llamado **Añadir controles para visualizar el elemento EGL en la página Web** y pulse **Aceptar**. Ahora el campo está disponible en la vista Datos de página. Puede añadirlo posteriormente al archivo JSP arrastrándolo de la vista Datos de página al JSP.

9. Si desea incluir los campos en el archivo JSP, siga estos pasos adicionales:
10. Seleccione el recuadro de selección llamado **Añadir controles para visualizar el elemento EGL en la página Web** y pulse **Aceptar**. Se abre la ventana Insertar control.
11. En la ventana Insertar control, marque el botón de selección que indica el uso que se hará del campo:
 - Para salida (**Visualizar un registro existente**)
 - Para entrada o salida (**Actualizar un registro existente**)
 - Para entrada (**Crear un registro nuevo**)
 La opción elegida afecta a los tipos de controles disponibles.
12. Para cambiar la etiqueta de campo, seleccione la etiqueta visualizada junto al nombre del campo y teclee el contenido nuevo.
13. Para seleccionar un tipo de control distinto del identificado, seleccione un tipo de la lista **Tipo de control**.
14. Si pulsa **Opciones**, se visualiza el diálogo Opciones y las opciones específicas disponibles dependen de si está utilizando el campo para entrada, para salida o para ambos casos. En cualquier caso, una opción consiste en incluir o excluir el código JSF `<h:outputLabel>` alrededor de las etiquetas de campo. Cuando finalice el trabajo en el diálogo Opciones, pulse **Aceptar**.
15. Pulse en **Finalizar**.

Consulta relacionada

“Soporte de Page Designer para EGL” en la página 192
 “Tipos primitivos” en la página 34

Asociar un registro EGL con un JSP Faces

Para asociar un registro EGL con un JSP Faces, haga lo siguiente:

1. Abra un archivo JSP Faces en Page Designer. Si no tiene un archivo JSP abierto, en el Explorador de proyectos pulse dos veces en el archivo JSP. El archivo JSP se abre en Page Designer. Pulse en la pestaña **Diseño** para acceder a la vista Diseño.

Nota: Puede acceder al PageHandler relacionado pulsando con el botón derecho del ratón en la vista Diseño (o en la vista Fuente) y pulsando **Editar código de página**.
2. En el menú **Ventana**, seleccione **Mostrar vista > Otras > Básica > Paleta**.
3. En la vista Paleta, pulse en **EGL** para visualizar los tipos de objetos de datos de EGL.
4. Arrastre **Registro** desde la paleta a la página JSP. Se visualizará el diálogo Seleccionar un componente de registro.
5. Seleccione un registro de la lista.
6. Especifique el nombre del campo o acepte el valor por omisión, que es el nombre del componente-registro.
7. Si está declarando una matriz de registros, marque el recuadro de selección **Matriz** y teclee un entero en el recuadro de selección **Tamaño**.
8. Si no desea incluir el registro en la página, quite la marca del recuadro de selección llamado **Añadir controles para visualizar el elemento EGL en la página Web** y pulse **Aceptar**. Ahora el registro está disponible en la vista Datos de página. Puede añadirlo posteriormente al archivo JSP arrastrándolo de la vista Datos de página al JSP.

9. Si desea incluir los campos en el archivo JSP, siga estos pasos adicionales:
10. Marque el recuadro de selección **Añadir controles para visualizar el elemento EGL en la página Web** y pulse **Aceptar**. Se abre la ventana Insertar control.
11. En la ventana Insertar control, marque el botón de selección que indica el uso que se hará del campo:
 - Para salida (**Visualizar un registro existente**)
 - Para entrada o salida (**Actualizar un registro existente**)
 - Para entrada (**Crear un registro nuevo**)La opción elegida afecta a los tipos de controles disponibles.
12. Para cambiar el orden de los campos, utilice las flechas arriba y abajo.
13. Si desea seleccionar solamente un subconjunto de los campos listados, pulse **Ninguno** y seleccione los campos deseados. Para seleccionar todos los campos, pulse **Todo**.
14. Haga lo siguiente para cada campo:
 - a. Para excluir un campo, quite la marca del recuadro de selección relacionado. Para incluir el campo, asegúrese de marcar el recuadro de selección.
 - b. Para cambiar la etiqueta de campo, seleccione la etiqueta visualizada junto al nombre del campo y teclee el contenido nuevo.
 - c. Para seleccionar un tipo de control distinto del identificado (si es posible), seleccione de una lista de tipos.
15. Si pulsa **Opciones**, se visualiza el diálogo Opciones y las opciones específicas disponibles dependen de si está utilizando campos para entrada, para salida o para ambos casos. En cualquier caso, una opción consiste en incluir o excluir el código JSF `<h:outputLabel>` alrededor de las etiquetas de campo.
Cuando finalice el trabajo en el diálogo Opciones, pulse **Aceptar**.
16. Pulse en **Finalizar**.

Conceptos relacionados

“Componentes de registro” en la página 132

Consulta relacionada

“Soporte de Page Designer para EGL” en la página 192

Enlazar componente mandato JavaServer Faces con PageHandler EGL

Para enlazar un componente de mandato de JavaServer Faces (botón o enlace de hipertexto) con una función de PageHandler de EGL, haga lo siguiente:

1. Abra un archivo JSP Faces en Page Designer. Si no tiene un archivo JSP abierto, en el Explorador de proyectos pulse dos veces en el archivo JSP. El JSP se abre en Page Designer. Pulse en la pestaña **Diseño** para acceder a la vista Diseño.
2. En el menú **Ventana**, seleccione **Mostrar vista > Otras > Básica > Paleta**.
3. En la vista Paleta, pulse la bandeja **Componentes Faces** para visualizar los tipos de objeto de Componentes Faces.
4. Arrastre un componente de mandato de la paleta al JSP. Los componentes de mandato incluyen la palabra **Mandato** en la etiqueta. El objeto de componente se coloca en el JSP.
5. Enlace un manejador de eventos con el componente de mandato utilizando uno de estos métodos:

- Para enlazar el componente con un manejador de eventos existente, arrastre el manejador de eventos desde el nodo Acciones de la vista Datos de página al objeto de componente en el JSP.
- Para crear un manejador de eventos nuevo enlazado al componente:
 - a. Pulse con el botón derecho del ratón sobre el componente y pulse **Editar eventos** en el menú emergente.
 - b. Utilizando la vista Edición rápida, especifique el código de EGL para el manejador de eventos. Para conocer los detalles sobre la utilización de la vista Edición rápida, consulte la sección *Utilizar la vista Edición rápida para el código de PageHandler*.

El manejador de eventos se muestra en la vista Datos de página y se añade una función correspondiente al PageHandler. Para conocer detalles, consulte el apartado *Componente PageHandler en formato fuente EGL*.

Conceptos relacionados

“PageHandler” en la página 194

Tareas relacionadas

“Crear un componente EGL pageHandler” en la página 191

“Utilizar la vista Edición rápida para el código de PageHandler”

Consulta relacionada

“Soporte de Page Designer para EGL” en la página 192

“Componente PageHandler en formato fuente EGL” en la página 679

Utilizar la vista Edición rápida para el código de PageHandler

La vista Edición rápida le permite mantener código de PageHandler EGL para eventos de servidor JSP sin abrir el archivo de PageHandler. Para utilizar la vista Edición rápida, haga lo siguiente:

1. Abra un archivo JSP en Page Designer. Si no tiene un archivo abierto, en el Explorador de proyectos pulse dos veces en el archivo JSP. El JSP se abre en Page Designer. Pulse en la pestaña **Diseño** para acceder a la vista Diseño.
2. Pulse con el botón derecho del ratón en Page Designer y seleccione **Editar eventos**. Se abrirá la vista Edición rápida.
3. Siga estos pasos para mantener funciones del PageHandler para componentes de mandato:
 - a. Seleccione un componente de mandato en el JSP.
 - b. Si el componente de mandato ya tiene asociada una función de PageHandler, la función se visualiza en el editor de scripts (panel derecho) de la vista Edición rápida. Los cambios que realice en el código quedan reflejados en el PageHandler.
 - c. Para crear una función de PageHandler para el componente de mandato seleccionado, pulse en **Mandato** en el panel de eventos (panel izquierdo) de la vista Edición rápida y, a continuación, pulse en el editor de scripts (panel derecho) de la vista Edición rápida. Se visualiza la función. Teclee el código de PageHandler para la función.
4. Siga estos pasos para mantener la función onPageLoad:
 - a. Pulse dentro del JSP.
 - b. Pulse en **onPageLoad** en el panel de eventos (panel izquierdo) de la vista Edición rápida.

- c. La función `onPageLoad` se visualiza en el editor de scripts (panel derecho) de la vista Edición rápida. Los cambios que realice en el código quedan reflejados en el `PageHandler`.

Conceptos relacionados

“`PageHandler`” en la página 194

Consulta relacionada

“Componente `PageHandler` en formato fuente EGL” en la página 679

Enlazar componente entrada/salida JavaServer Faces con `PageHandler` EGL

Para enlazar un componente de entrada o salida de JavaServer Faces con un área de datos de `PageHandler` de EGL existente, haga lo siguiente:

1. Abra un archivo JSP Faces en Page Designer. Si no tiene un archivo JSP abierto, en el Explorador de proyectos pulse dos veces en el archivo JSP. El JSP se abre en Page Designer. Pulse en la pestaña **Diseño** para acceder a la vista Diseño.
2. En el menú **Ventana**, seleccione **Mostrar vista > Otras > Básica > Paleta**.
3. En la vista Paleta, pulse la bandeja **Componentes Faces** para visualizar los tipos de objeto de Componentes Faces.
4. Arrastre un componente de entrada o salida desde la paleta al JSP. Los componentes de entrada y salida tienen las palabras **Entrada** y **Salida** en las etiquetas. El objeto de componente se coloca en el JSP.
5. Para enlazar el componente con un área de datos de `PageHandler` existente, lleve a cabo una de las acciones siguientes:
 - Arrastre el área de datos desde la vista Datos de página al objeto de componente en el JSP.
 - Seleccione el objeto de componente en el JSP, pulse con el botón derecho del ratón el área de datos de la vista Datos de página y seleccione **Enlazar a 'nombre de componente'**.
 - Seleccione el objeto de componente en el JSP. Pulse el botón situado junto al campo **Valor** de la vista Propiedades, seleccione un área de datos de la lista Seleccionar objeto de datos de página y pulse **Aceptar**.

Conceptos relacionados

“`PageHandler`” en la página 194

Tareas relacionadas

“Crear un componente EGL `pageHandler`” en la página 191

Consulta relacionada

“Soporte de Page Designer para EGL” en la página 192

“Componente `PageHandler` en formato fuente EGL” en la página 679

Enlazar comp. recuadro selección JavaServer Faces con `PageHandler` EGL

Un componente de recuadro de selección de JavaServer Faces es exclusivo en cuanto a que el área de datos con la que está enlazado debe tener la propiedad de elemento **isBoolean** (antes la propiedad **boolean**) establecida en *yes*. A continuación se proporcionan ejemplos de declaraciones de área de datos booleanas:

```

DataItem CharacterBooleanItem char(1)
{
    value = "N",
    isBoolean = yes
}
end
DataItem NumericBooleanItem smallInt
{
    value = "0",
    isBoolean = yes
}
end

```

Para enlazar un componente de recuadro de selección de JavaServer Faces con un área de datos de PageHandler de EGL existente, haga lo siguiente:

1. Abra un archivo JSP Faces en Page Designer. Si no tiene un archivo JSP abierto, en el Explorador de proyectos pulse dos veces en el archivo JSP. El JSP se abre en Page Designer. Pulse en la pestaña **Diseño** para acceder a la vista Diseño.
2. En el menú **Ventana**, seleccione **Mostrar vista > Otras > Básica > Paleta**.
3. En la vista Paleta, pulse la bandeja **Componentes Faces** para visualizar los tipos de objeto de Componentes Faces.
4. Arrastre un componente de recuadro de selección de la paleta al JSP. El objeto de componente se coloca en el JSP.
5. Para enlazar el componente con un área de datos de PageHandler existente, lleve a cabo una de las acciones siguientes:
 - Arrastre el área de datos desde la vista Datos de página al objeto de componente en el JSP.
 - Seleccione el objeto de componente en el JSP, pulse con el botón derecho del ratón el área de datos de la vista Datos de página y seleccione **Enlazar a 'nombre de componente'**.
 - Seleccione el objeto de componente en el JSP. Pulse el botón situado junto al campo **Valor** en la vista Propiedades, seleccione un área de datos de la lista Seleccionar objeto de datos de página y pulse **Aceptar**.

Conceptos relacionados

“PageHandler” en la página 194

Tareas relacionadas

“Crear un componente EGL pageHandler” en la página 191

Consulta relacionada

“Soporte de Page Designer para EGL” en la página 192

“Componente PageHandler en formato fuente EGL” en la página 679

Enlazar componente selección única JavaServer Faces con PageHandler EGL

Un componente de selección única permite a un usuario hacer una selección de una lista de valores. La selección del usuario se almacena en un área de datos de PageHandler. Los botones de selección, los recuadros de lista de selección única y los cuadros combinados son componentes JavaServer Faces de selección única.

Un enlace es una relación entre el componente y un área de datos. El área de datos debe declararse en el PageHandler para poder enlazar un componente. Un componente de selección única necesita dos clases diferentes de enlaces:

- Un enlace a una o varias áreas de datos que contengan los valores entre los que el usuario puede realizar una selección
- Un enlace a un área de datos que recibirá la selección del usuario

Para enlazar un componente de selección única de JavaServer Faces con áreas de datos de PageHandler de EGL, haga lo siguiente:

1. Abra un archivo JSP Faces en Page Designer. Si no tiene un archivo JSP abierto, en el Explorador de proyectos pulse dos veces en el archivo JSP. El JSP se abre en Page Designer. Pulse en la pestaña **Diseño** para acceder a la vista Diseño.
2. En el menú **Ventana**, seleccione **Mostrar vista > Otras > Básica > Paleta**.
3. En la vista Paleta, pulse la bandeja **Componentes Faces** para visualizar los tipos de objeto de Componentes Faces.
4. Arrastre un componente de selección única de la paleta al JSP. El objeto de componente se coloca en el JSP.
5. Para enlazar el componente con una o varias áreas de datos de PageHandler que contienen los valores que desea mostrar al usuario, siga uno de los procedimientos siguientes:
 - Puede enlazar el componente con áreas de datos de PageHandler individuales, cada una de las cuales contiene un elemento de lista. Realice el procedimiento siguiente para cada área de datos:
 - a. Seleccione el componente de objeto en el JSP.
 - b. En la vista Propiedades, pulse **Añadir opción**. Los campos Nombre y Valor se pueblan con valores por omisión.
 - c. Pulse el campo **Nombre** y teclee el texto que debe mostrarse al usuario.
 - d. Pulse el campo **Valor** y después pulse el botón situado junto al campo **Valor**. Seleccione un área de datos individual en la lista Seleccionar objeto de datos de página y pulse **Aceptar**. Este área contiene el valor que se moverá al área de datos de recepción.
 - Puede enlazar el componente con un área de datos de matriz de PageHandler que contenga los valores que desea mostrar al usuario. Lleve a cabo el procedimiento siguiente para enlazar el componente con un área de datos de matriz:
 - a. Seleccione el componente de objeto en el JSP.
 - b. En la vista Propiedades, pulse **Añadir conjunto de opciones**. Los campos Nombre y Valor se pueblan con valores por omisión.
 - c. Pulse el campo **Valor** y después pulse el botón situado junto al campo **Valor**. Seleccione un área de datos de matriz en la lista Objeto de datos de página y pulse **Aceptar**. Los valores del área de datos de matriz son los valores que se mostrarán al usuario. Las propiedades que se describen más adelante determinan si los valores del área de datos de matriz o los valores de índice equivalentes se moverán al área de datos de recepción.
6. Si está utilizando un área de datos para proporcionar los valores mostrados al usuario debe definir el área de datos de recepción con dos propiedades: **selectFromListItem** y **selectType**. La propiedad **selectFromListItem** señala la matriz que alberga los elementos de lista. La propiedad **selectType** indica si el área de datos de recepción debe poblarse con un valor de texto o un valor de índice. A continuación se proporcionan ejemplos de áreas de datos de recepción:

```
colorSelected char(10)
{selectFromListItem = "colorsArray",
selectType = value};
```

```
colorSelectIdx smallInt
{selectFromListItem = "colorsArray",
selectType = index};
```

7. Para enlazar el componente con un área de datos de PageHandler que recibirá la selección del usuario, lleve a cabo una de las acciones siguientes:
 - Arrastre el área de datos desde la vista Datos de página al objeto de componente en el JSP.
 - Seleccione el objeto de componente en el JSP, pulse con el botón derecho del ratón el área de datos de la vista Datos de página y seleccione **Enlazar a 'nombre de componente'**.
 - Seleccione el objeto de componente en el JSP. Pulse el botón situado junto al campo **Valor** en la vista Propiedades, seleccione un área de datos de la lista Seleccionar objeto de datos de página y pulse **Aceptar**.

Conceptos relacionados

"PageHandler" en la página 194

Tareas relacionadas

"Crear un componente EGL pageHandler" en la página 191

Consulta relacionada

"Soporte de Page Designer para EGL" en la página 192

"Componente PageHandler en formato fuente EGL" en la página 679

Enlazar comp. selección múltiple JavaServer Faces con PageHandler EGL

Un componente de selección múltiple permite a un usuario hacer una o varias selecciones de una lista de valores. Las selecciones del usuario se almacenan en un área de datos de matriz de PageHandler. Los grupos de recuadro de selección y los cuadros de lista de varias selecciones son componentes JavaServer Faces de selección múltiple.

Un enlace es una relación entre el componente y un área de datos. El área de datos debe declararse en el PageHandler para poder enlazar un componente. Un componente de selección múltiple necesita dos clases diferentes de enlaces:

- Un enlace a una o varias áreas de datos que contengan los valores entre los que el usuario puede realizar una selección
- Un enlace a un área de datos de matriz que recibirá las selecciones del usuario

Para enlazar un componente de selección múltiple de JavaServer Faces con áreas de datos de PageHandler de EGL, haga lo siguiente:

1. Abra un archivo JSP Faces en Page Designer. Si no tiene un archivo JSP abierto, en el Explorador de proyectos pulse dos veces en el archivo JSP. El JSP se abre en Page Designer. Pulse en la pestaña **Diseño** para acceder a la vista Diseño.
2. En el menú **Ventana**, seleccione **Mostrar vista > Otras > Básica > Paleta**.
3. En la vista Paleta, pulse la bandeja **Componentes Faces** para visualizar los tipos de objeto de Componentes Faces.
4. Arrastre un componente de selección múltiple de la paleta al JSP. El objeto de componente se coloca en el JSP.
5. Para enlazar el componente con una o varias áreas de datos de PageHandler que contienen los valores que desea mostrar al usuario, siga uno de los procedimientos siguientes:

- Puede enlazar el componente con áreas de datos de PageHandler individuales, cada una de las cuales contiene un elemento de lista. Realice el procedimiento siguiente para cada área de datos:
 - a. Seleccione el componente de objeto en el JSP.
 - b. En la vista Propiedades, pulse **Añadir opción**. Los campos Nombre y Valor se pueblan con valores por omisión.
 - c. Pulse el campo **Nombre** y teclee el texto que debe mostrarse al usuario.
 - d. Pulse el campo **Valor** y después pulse el botón situado junto al campo **Valor**. Seleccione un área de datos individual en la lista Seleccionar objeto de datos de página y pulse **Aceptar**. Este área contiene el valor que se moverá al área de datos de recepción.
 - Puede enlazar el componente con un área de datos de matriz de PageHandler que contenga los valores que desea mostrar al usuario. Lleve a cabo el procedimiento siguiente para enlazar el componente con un área de datos de matriz:
 - a. Seleccione el componente de objeto en el JSP.
 - b. En la vista Propiedades, pulse **Añadir conjunto de opciones**. Los campos Nombre y Valor se pueblan con valores por omisión.
 - c. Pulse el campo **Valor** y después pulse el botón situado junto al campo **Valor**. Seleccione un área de datos de matriz en la lista Seleccionar objeto de datos de página y pulse **Aceptar**. Los valores del área de datos de matriz son los valores que se mostrarán al usuario. Las propiedades que se describen más adelante determinan si los valores del área de datos de matriz o los valores de índice equivalentes se moverán al área de datos de recepción.
6. Si está utilizando un área de datos para proporcionar los valores mostrados al usuario debe definir el área de datos de recepción con dos propiedades: **selectFromListItem** y **selectType**. La propiedad **selectFromListItem** señala la matriz que alberga los elementos de lista. La propiedad **selectType** indica si el área de datos de recepción debe poblarse con un valor de texto o un valor de índice. A continuación se proporcionan ejemplos de áreas de datos de recepción:
- ```
colorSelected char(10)
{selectFromListItem = "colorsArray",
 selectType = value};

colorSelectIdx smallInt
{selectFromListItem = "colorsArray",
 selectType = index};
```
7. Para enlazar el componente con un área de datos de matriz de PageHandler que recibirá las selecciones del usuario, haga lo siguiente:
- a. Seleccione el objeto de componente en el JSP
  - b. Pulse el botón situado junto al campo **Valor** de la vista Propiedades
  - c. Seleccione un área de datos de la lista Seleccionar objeto de datos de página
  - d. Pulse **Aceptar**

#### Conceptos relacionados

“PageHandler” en la página 194

#### Tareas relacionadas

“Crear un componente EGL pageHandler” en la página 191

**Consulta relacionada**

“Soporte de Page Designer para EGL” en la página 192

“Componente PageHandler en formato fuente EGL” en la página 679

---

## Crear informes de EGL

---

### Visión general de los informes de EGL

EGL puede producir informes basados en las funciones de JasperReports, que es una biblioteca de informes de código fuente abierto basada en Java. Para obtener detalles acerca de dicha biblioteca, consulte el siguiente sitio Web:

<http://jasperreports.sourceforge.net>

EGL no suministra ningún mecanismo para el diseño de informes. Debe proceder del siguiente modo:

- Importe un archivo de salida JasperReports (extensión de archivo *jasper*); o
- Utilice un editor de texto o una herramienta especializada para crear un archivo de diseño que se transformará en un archivo de salida JasperReports cuando pulse **Proyecto > Construir todo** en el Entorno de trabajo.

A continuación figuran dos herramientas especializadas para crear un archivo de diseño:

- JasperAssistant, que se describe en este sitio Web:

<http://www.jasperassistant.com>

- iReport, que se describe en este sitio Web:

<http://ireport.sourceforge.net>

En el archivo EGL que escriba para realizar la producción de informes, someterá los datos al archivo de salida de JasperReports (o aceptará el origen de datos especificado en dicho archivo) y, a continuación, exportará el informe a uno o varios archivos de salida, cada uno de los cuales puede estar en un formato diferente, como por ejemplo HTML o Adobe Acrobat PDF.

Si también codifica un manejador EGL de tipo JasperReport, puede responder a los eventos de usuario que se produzcan cuando el informe se rellena con datos; por ejemplo, puede añadir detalles específicos de ejecución al informe cuando la producción del informe casi haya finalizado. Para asegurarse de que el manejo de eventos funciona, sin embargo, debe asegurarse de que se hace referencia a la salida generada desde el manejador de informes en el archivo de salida de JasperReports.

El **asistente Manejador de informes EGL** permite crear con facilidad un manejador de informes EGL.

Cuando escriba código EGL que interactúe con un informe, utilizará las funciones de la biblioteca de sistema **ReportLib**; y, en el código que produce un informe, creará variables de tipo Report y ReportData.

Los componentes de EGL mencionados aquí (Handler, Report y Report Data) se definen automáticamente.

#### Conceptos relacionados

Visión general del proceso de creación de informes de EGL

**Consulta relacionada**  
Biblioteca de informes de EGL  
Orígenes de datos  
Manejador de informes de EGL

---

## Visión general del proceso de creación de informes de EGL

En este tema se proporciona una visión general de los procesos para crear y generar un informe para un proyecto EGL. Encontrará más detalles acerca de estos procesos en los temas de ayuda de las tareas de informes de EGL.

Para crear un informe, puede completar los tres procesos descritos a continuación. Dos de estos procesos, la creación de un diseño XML y la escritura de código para controlar un informe, son obligatorios. Un tercer proceso, la creación de un manejador de informes, es opcional. No es necesario que realice estos procesos en el orden descrito. Por ejemplo, si desea un manejador de informes, puede crearlo antes de crear un documento de diseño XML o pueden trabajar simultáneamente en la creación de un documento de diseño y un manejador de informes, con las excepciones descritas en los párrafos de "Interrelaciones de código entre el manejador de informes y el documento de diseño XML" del paso 2.

No puede generar un informe si no tiene un documento de diseño XML ni el código para controlar el informe.

Los tres procesos que hay que realizar para crear un informe son:

1. Crear un documento de diseño XML para especificar información de diseño para el informe. Puede crear este documento de cualquiera de las formas siguientes:
  - Utilizando una herramienta de diseño de JasperReport de terceros (como por ejemplo Jasper Assistant o iReports).
  - Utilizando un editor de texto para escribir información de diseño XML de Jasper en un archivo de texto nuevo.

El documento de diseño XML debe tener una extensión .jrxml. Si el archivo creado no tiene esta extensión, redemóinelo como archivo .jrxml. Además, asegúrese de que el documento de diseño XML esté en el mismo paquete EGL que contendrá el manejador de informes EGL y los archivos de código de invocación de informe.

El archivo .jrxml creado se compilará en un archivo .jasper. Si no crea un archivo .jrxml nuevo, debe importar un archivo .jasper que se haya compilado con anterioridad.
2. Si desea utilizar un manejador de informes que proporciona la lógica para manejar sucesos durante la confección del informe, puede crear un manejador de informes de cualquiera de las formas siguientes:
  - Utilizando el asistente de manejador de informes de EGL para especificar información para el manejador de informes.
  - Creando un archivo fuente EGL nuevo e insertando un manejador mediante la plantilla de manejador de informes o especificando manualmente el código del manejador.

**Interrelaciones de código entre el manejador de informes y el documento de diseño XML.** En el archivo .jrxml, puede especificar la scriptletClass que hace referencia al archivo de manejador de informes generado por el manejador de informes de EGL. Tenga en cuenta lo siguiente:

- Si el archivo .jrxml utiliza código Java generado por un manejador de informes, debe generar el manejador de informes antes de crear el archivo .jrxml.
  - Si cambia un manejador de informes, debe volver a compilar el archivo .jrxml.
  - Si necesita resolver cualesquiera errores de compilación en el archivo .jrxml o desea volver a compilar el archivo .jasper después de hacer cambios en un manejador de informes, debe modificar el archivo .jrxml y guardarlo.
3. Utilice las funciones ReportLib de EGL para escribir código de invocación de informes en el proyecto EGL. Puede utilizar el asistente Componente de programa EGL al crear código de invocación de informes.

**Importante:** debe dar nombres de archivo al manejador de informes y al código de invocación de informe que son distintos del nombre del documento de diseño XML. Si no lo hace, al compilar el archivo de diseño se sobrescribirá el código Java. Para evitar problemas, llame a los manejadores de informes *reportName\_handler.egl* y a los documentos de diseño XML *reportName\_XML.jrxml*. Por ejemplo, puede llamar al informe *abc\_handler.egl* y a los documentos de diseño *abc\_XML.jrxml*. También debe asegurarse de que el archivo de diseño XML tenga un nombre exclusivo de forma que no entre en conflicto con ninguno de los archivos de programa EGL.

Para construir y generar un informe después de crear un documento de diseño XML, un manejador de informes si desea utilizar uno y código de invocación de informe, debe realizar los procesos siguientes:

1. Construya el proyecto EGL seleccionando **Proyecto > Construir todo**.  
EGL genera automáticamente código Java a partir del manejador de informes EGL y compila el documento de diseño XML (el archivo .jrxml) en un archivo .jasper.
2. Ejecute el programa EGL que tiene el código de invocación de informe.

Una vez ejecutado el programa EGL, el programa JasperReports utilizado por EGL guarda automáticamente el informe generado en la ubicación especificada por *reportDestinationFileName* en el código de invocación de informes.

El programa JasperReports que genera el informe también genera y almacena un archivo .jprint que es un formato de archivo intermedio que se exporta al formato de informe final (.pdf, .html, .xml, .txt o .csv).

El programa puede reutilizar un .jprint para varias exportaciones.

La función `exportReport()` del código de invocación de informes hace que EGL exporte el informe en el formato especificado. Por ejemplo, el código siguiente hace que EGL exporte un informe en formato .pdf:

```
reportLib.exportReport(myReport, ExportFormat.pdf);
```

EGL no renueva automáticamente informes exportados. Si cambia el diseño del informe o si los datos cambian, deberá volver a confeccionar y a exportar el informe.

### Conceptos relacionados

Visión general del informe EGL

### Tareas relacionadas

Añadir un documento de diseño a un paquete  
Utilizar plantillas de informe  
Crear un manejador de informes de EGL  
Crear manualmente un manejador de informes de EGL  
Escribir código para controlar un informe  
Ejecutar un informe  
Exportar informes  
Utilizar la ayuda de contenido en EGL

**Consulta relacionada**

Biblioteca de informes de EGL  
Orígenes de datos  
Manejador de informes de EGL

---

## Orígenes de datos

La biblioteca de informes de EGL incluye referencias al origen de datos primario que contiene datos o a información acerca de cómo obtener los datos.

Puede utilizar las sentencias siguientes para especificar información de origen de datos:

- *DataSource.databaseConnection*
- *DataSource.sqlStatement*
- *DataSource.reportData*

Por ejemplo, si especifica *fillReport (eglReport, DataSource.databaseConnection)* al utilizar la función *ReportLib.fillReport*, EGL recupera la conexión de base de datos y la pasa al motor JasperReports.

Consulte Código de ejemplo para funciones de controlador de informes de EGL para obtener ejemplos de generación de informes mediante una conexión de base de datos, datos de informe y una sentencia SQL como origen de datos.

**Conceptos relacionados**

“Visión general de los informes de EGL” en la página 209  
“Visión general del proceso de creación de informes de EGL” en la página 210

**Consulta relacionada**

“Biblioteca ReportLib de EGL” en la página 861  
“fillReport()” en la página 863  
“Código de ejemplo para funciones de controlador de informes de EGL” en la página 217

---

## Registros de datos en la biblioteca

La biblioteca EGL contiene un registro *ReportData* y un registro *Report*.

El registro *ReportData* contiene información acerca de un conjunto de datos determinado que se va a utilizar en un informe. El registro contiene estos campos:

| Campo                 | Descripción                                           | Tipo de datos |
|-----------------------|-------------------------------------------------------|---------------|
| <i>connectionName</i> | Nombre de la conexión establecida en el programa EGL. | String        |

| Campo               | Descripción                                                                                                        | Tipo de datos                       |
|---------------------|--------------------------------------------------------------------------------------------------------------------|-------------------------------------|
| <i>sqlStatement</i> | La sentencia SQL que debe ejecutar EGL. Los datos de informe vienen del resultado de la ejecución de la sentencia. | String                              |
| <i>Data</i>         | Una matriz dinámica de registros.                                                                                  | Any (es el tipo <i>Any</i> de EGL.) |

El registro *Report* contiene información de un registro determinado. El registro contiene estos campos:

| Campo                        | Descripción                                                                                                  | Tipo de datos |
|------------------------------|--------------------------------------------------------------------------------------------------------------|---------------|
| <i>reportDesignFile</i>      | Nombre del archivo de diseño de informes, que es un archivo XML con una extensión <i>.jrxml</i> .            | String        |
| <i>reportDestinationFile</i> | Ubicación del archivo <i>.jrprint</i> .                                                                      | String        |
| <i>reportExportFile</i>      | Ubicación del archivo final, guardado <i>.xml</i> , <i>.pdf</i> , <i>.html</i> , <i>.txt</i> o <i>.csv</i> . | String        |
| <i>reportData</i>            | Los datos de informe que se utilizan como el origen de datos primario del informe.                           |               |

#### Conceptos relacionados

“Visión general de los informes de EGL” en la página 209

“Visión general del proceso de creación de informes de EGL” en la página 210

#### Consulta relacionada

“Biblioteca ReportLib de EGL” en la página 861

---

## Manejador de informes de EGL

El manejador de informes de EGL proporciona funciones adicionales para manejar eventos que se producen cuando se rellena un informe con datos. Puede utilizar el asistente Manejador de informes EGL nuevo para especificar información para un manejador de informes o puede crear uno manualmente.

Cuando se genera un archivo manejador de informes, EGL crea estos archivos:

- *handlerName.java*
- *handlerName\_lib.java*.

*handlerName*

Alias del manejador de informes de EGL

Cuando EGL genera archivos *.java*, los nombres de clase están en minúsculas. Asegúrese de cualquier nombre de clase especificado en un documento de diseño XML esté en minúsculas.

Consulte el apartado Crear manualmente un manejador de informes de EGL para obtener ejemplos de la sintaxis y de código de manejador de informes.

**Detalles técnicos:** el manejador de informes de EGL es un componente manejador de EGL de tipo JasperReport. El manejador de informes se correlaciona con la clase de scriptlet JasperReports. La generación del manejador de informes Java amplía la clase JRDefaultScriptlet y define una clase Java que contiene las funciones Java generadas que representan las funciones de scriptlet. La sección de definición del documento de diseño XML contiene el nombre de la clase de scriptlet. El motor JasperReports carga la clase de scriptlet y llama a distintos métodos tal como se especifica en la definición del informe. (Para obtener más información sobre scriptlets de JasperReports y clases de scriptlet, consulte la documentación de JasperReports.)

El manejador de informes mantiene una lista interna de registros *ReportData* que se devuelven cuando se solicita.

#### Conceptos relacionados

“Visión general de los informes de EGL” en la página 209

“Visión general del proceso de creación de informes de EGL” en la página 210

#### Tareas relacionadas

“Migrar código EGL a EGL 6.0 iFix” en la página 101

“Crear manualmente un manejador de informes de EGL” en la página 221

“Escribir código para controlar un informe” en la página 225

#### Consulta relacionada

“Funciones de manejador de informes de EGL adicionales” en la página 215

“Biblioteca ReportLib de EGL” en la página 861

“Funciones de manejador de informes predefinidas”

---

## Funciones de manejador de informes predefinidas

El manejador de informes proporciona las funciones predefinidas siguientes que puede utilizar como plantillas de función:

| Función                                      | Dónde opera la función                                                                                   |
|----------------------------------------------|----------------------------------------------------------------------------------------------------------|
| beforeReportInit();                          | Antes de la inicialización de informes                                                                   |
| afterReportInit();                           | Después de la inicialización de informes                                                                 |
| beforePageInit();                            | Al entrar en una página                                                                                  |
| afterPageInit();                             | Al abandonar una página                                                                                  |
| beforeColumnInit();                          | Antes de la inicialización de columnas                                                                   |
| afterColumnInit();                           | Después de la inicialización de columnas                                                                 |
| beforeGroupInit ( <i>groupName String</i> ); | Antes de la inicialización de grupo.<br><i>groupName</i> es el nombre del grupo en el informe.           |
| afterGroupInit( <i>groupName String</i> );   | Después de la inicialización de grupo.                                                                   |
| beforeDetailEval();                          | Antes de cada campo. Si se establece esta función, cada fila, antes de imprimir, llama a esta función.   |
| afterDetailEval();                           | Después de cada campo. Si se establece esta función, cada fila, antes de imprimir, llama a esta función. |

Dentro de una de estas funciones puede hacer llamadas a otras funciones. Por ejemplo, puede hacer una llamada a `setReportVariable()`, de la forma siguiente:

```
function afterGroupInit(groupName String)
 if (groupName == "cat")
 setReportVariableValue ("NewGroupName", "dog");
 else
 setReportVariableValue ("NewGroupName", groupName);
 end
end
```

También puede crear sus propias funciones. Consulte la documentación de JasperReports para obtener información acerca de la creación de funciones personalizadas.

Para ver ejemplos sobre cómo utilizar las funciones predefinidas del manejador de informes, consulte el apartado *Crear manualmente un manejador de informes de EGL*.

#### Conceptos relacionados

“Visión general de los informes de EGL” en la página 209

“Visión general del proceso de creación de informes de EGL” en la página 210

#### Tareas relacionadas

“Migrar código EGL a EGL 6.0 iFix” en la página 101

“Crear manualmente un manejador de informes de EGL” en la página 221

#### Consulta relacionada

“Funciones de manejador de informes de EGL adicionales”

“Registros de datos en la biblioteca” en la página 212

“Biblioteca ReportLib de EGL” en la página 861

“Manejador de informes de EGL” en la página 213

---

## Funciones de manejador de informes de EGL adicionales

Puede invocar cualquiera de las funciones siguientes de ReportLib desde las funciones de manejador de informes predefinidas:

#### Función para obtener parámetros de informe

| Función                                                        | Finalidad                                                                            |
|----------------------------------------------------------------|--------------------------------------------------------------------------------------|
| <code>getReportParameter</code> ( <i>parámetro String in</i> ) | Devuelve el valor del parámetro especificado del informe que se está cumplimentando. |

#### Funciones para establecer y obtener variables de informe

Estas variables pueden utilizarse por muchas razones, por ejemplo, para almacenar una expresión utilizada frecuentemente o para realizar un cálculo complejo en la expresión definida en la fila que se procesa.

| Función                                                                          | Finalidad                                                                                                               |
|----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <code>getReportVariableValue</code> ( <i>variable String in</i> )                | Devuelve el valor de la variable especificada del informe que se está cumplimentando. El valor devuelto es de tipo ANY. |
| <code>setReportVariableValue</code> ( <i>variable String in, valor Any in</i> ); | Establece el valor de la variable especificada en el valor proporcionado.                                               |

### Función para obtener valores de campo

| Función                                                                   | Finalidad                                                                                                                   |
|---------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <code>getFieldValue</code> ( <i>nombreCampo</i> <b>String</b> <u>in</u> ) | Devuelve el valor del campo especificado para la fila que se está procesando actualmente. El valor devuelto es de tipo ANY. |

### Funciones para añadir u obtener datos para subinformes

Un subinforme es un informe que se llama desde otro informe. A veces se intercambian datos entre el informe principal y el subinforme. Un subinforme también puede ser el informe principal de otro subinforme.

| Función                                                                                                               | Finalidad                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| <code>addReportData</code> ( <i>rd datosInforme</i> <u>in</u> , <i>nombreConjuntoDatos</i> <b>String</b> <u>in</u> ); | Añade el objeto de datos de informe con el nombre especificado al manejador de informes actual.               |
| <code>getReportData</code> ( <i>nombreConjuntoDatos</i> <b>String</b> <u>in</u> )                                     | Recupera el registro de datos de informe con el nombre especificado. El valor devuelto es de tipo ReportData. |

Para ver ejemplos sobre cómo utilizar las funciones descritas en este tema, consulte el apartado *Crear manualmente un manejador de informes de EGL*.

#### Conceptos relacionados

“Visión general del proceso de creación de informes de EGL” en la página 210

“Visión general de los informes de EGL” en la página 209

#### Tareas relacionadas

“Migrar código EGL a EGL 6.0 iFix” en la página 101

“Crear manualmente un manejador de informes de EGL” en la página 221

#### Consulta relacionada

“Registros de datos en la biblioteca” en la página 212

“Manejador de informes de EGL” en la página 213

“Biblioteca ReportLib de EGL” en la página 861

“Funciones de manejador de informes predefinidas” en la página 214

---

## Tipos de datos en documentos de diseño XML

En los documentos de diseño de informes XML, los tipos de datos se describen como tipos de datos Java. Si crea código de scriptlet de EGL para utilizarlo en el documento de diseño, utilice el tipo de datos Java que se corresponda con el tipo de datos primitivo de EGL aplicable. Los datos devueltos como resultado de una llamada al código de scriptlet de EGL deben declararse utilizando el tipo de datos Java.

La tabla siguiente muestra cómo se correlacionan los tipos de datos primitivos de EGL con los tipos de datos Java. La documentación de JavaReports contiene información acerca de los tipos de datos Java que puede utilizar.

| Tipo primitivo EGL | Tipo de datos Java   |
|--------------------|----------------------|
| bigint             | java.lang.Long       |
| bin                | java.math.BigDecimal |
| blob               |                      |
| char               | java.lang.String     |
| clob               |                      |
| date               | java.util.Date       |
| dbchar             | java.lang.String     |
| decimal            | java.math.BigDecimal |
| decimalfloat       | java.lang.Double     |
| float              | java.lang.Float      |
| hex                | java.lang.byte       |
| int                | java.lang.Integer    |
| interval           | java.lang.String     |
| mbchar             | java.lang.String     |
| money              | java.math.BigDecimal |
| numc               | java.math.BigDecimal |
| pacf               | java.math.BigDecimal |
| smallfloat         | java.lang.Float      |
| smallint           | java.lang.Short      |
| string             | java.lang.String     |
| time               | java.sql.Time        |
| timestamp          | java.sql.Timestamp   |
| unicode            | java.lang.String     |

#### Conceptos relacionados

“Visión general de los informes de EGL” en la página 209

“Visión general del proceso de creación de informes de EGL” en la página 210

#### Tareas relacionadas

“Añadir un documento de diseño a un paquete” en la página 219

#### Consulta relacionada

“Registros de datos en la biblioteca” en la página 212

“Biblioteca ReportLib de EGL” en la página 861

“Manejador de informes de EGL” en la página 213

---

## Código de ejemplo para funciones de controlador de informes de EGL

Este tema contiene fragmentos de código que muestran cómo se genera un informe utilizando tres orígenes de datos distintos:

- Una conexión de base de datos
- Un registro de datos
- Una sentencia SQL

El fragmento de código siguiente muestra cómo se genera un informe utilizando una conexión de base de datos como el origen de datos:

```
//Declaración de variable
myReport Report;
myReportData ReportData;

//Función que contiene código de invocación de informes
function makeReport()
 //Inicializar ubicaciones de archivo de informe
 myReport.reportDesignFile = "reportDesignFileName.jasper";
 myReport.reportDestinationFile =
"reportDestinationFileName.jrprint";

 //Establecer los datos de informe a través de una conexión con la sentencia SQL
 //incorporada en el diseño de informe
 sysLib.defineDatabaseAlias("alias", "databaseName");
 sysLib.connect("alias", "userid", "password");
 myReportData.connectionName="connectionName";
 myReport.reportData = myReportData;

 //Cumplimentar el informe con datos
 reportLib.fillReport(myReport, DataSource.databaseConnection);
 //Exportar el informe en formato PDF
 myReport.reportExportFile = "reportDesignFileName.pdf";
 reportLib.exportReport(myReport, ExportFormat.pdf);
end
```

El fragmento de código siguiente muestra cómo se genera un informe utilizando un registro flexible de datos como el origen de datos:

```
//Declaración de variable
myReport Report;
myReportData ReportData;

//Función que contiene el código de control de informe
function makeReport()
 //Inicializar ubicaciones de archivo de myReport
 myReport.reportDesignFile = "reportDesignFileName.jasper";
 myReport.reportDestinationFile =
"reportDestinationFileName.jrprint";

 //Establecer los datos de informe
 populateReportData();
 myReport.reportData = myReportData;

 //Cumplimentar el informe con datos
 reportLib.fillReport(myReport, DataSource.reportData);

 //Exportar el informe en formato HTML
 myReport.reportExportFile = "reportDesignFileName.html";
 reportLib.exportReport(myReport, ExportFormat.html);
end

function populateReportData()
 //Insertar código EGL aquí que pueble myReportData
 ...
end
```

El fragmento de código siguiente muestra cómo se genera un informe utilizando una sentencia SQL como el origen de datos:

```
//Declaración de variable
myReport Report;
myReportData ReportData;
```

```

//Función que contiene código de control de informe
function makeReport()
 //Inicializar ubicaciones de archivo de informe
 myReport.reportDesignFile = "reportDesignFileName.jasper";
 myReport.reportDestinationFile = "reportDestinationFileName.jrprint";

 //Establecer los datos de informe a través de una sentencia SQL
 myReportData.sqlStatement = "SELECT * FROM dataBaseTable";
 myReport.reportData = myReportData;

 //Cumplimentar el informe con datos
 reportLib.fillReport(myReport, DataSource.sqlStatement);

 //Exportar el informe en formato de texto
 myReport.reportExportFile = "reportOutputFileName.txt";
 reportLib.exportReport(myReport, ExportFormat.text);
end

```

### Conceptos relacionados

“Visión general del proceso de creación de informes de EGL” en la página 210

“Visión general de los informes de EGL” en la página 209

### Tareas relacionadas

“Escribir código para controlar un informe” en la página 225

### Consulta relacionada

“Registros de datos en la biblioteca” en la página 212

“Orígenes de datos” en la página 212

“Manejador de informes de EGL” en la página 213

“Biblioteca ReportLib de EGL” en la página 861

---

## Añadir un documento de diseño a un paquete

Debe tener un documento de diseño XML con una extensión .jrxml que especifique la información de diseño del informe.

Para añadir un documento de diseño a un paquete, siga estos pasos:

1. Cree un documento de diseño de cualquiera de las maneras siguientes:
  - Utilice una herramienta de diseño JasperReports de terceros (como por ejemplo JasperAssistant o iReports). Si el archivo que crea no tiene una extensión .jrxml, redénomine el archivo para que tenga una extensión .jrxml.
  - Utilice un editor de texto para escribir información de diseño XML de Jasper XML en un archivo de texto nuevo y guarde el archivo como un archivo .jrxml.
2. Sitúe el documento de diseño XML en el mismo paquete EGL que albergará el manejador de informes EGL y los archivos de controlador de informes.

Si no crea un documento de diseño XML nuevo, debe importar un archivo .jasper que se haya compilado anteriormente.

El archivo .jrxml se compila automáticamente en un archivo .jasper cuando selecciona **Proyecto > Construir todo** para construir todos los componentes de proyecto EGL.

**Nota:** Consulte el apartado Visión general del proceso de creación de informes de EGL para conocer las directrices a seguir para crear simultáneamente un documento de diseño XML y un manejador de informes. Consulte el apartado Crear manualmente un componente de manejador de informes

EGL para obtener un ejemplo que muestre cómo un documento de diseño XML obtiene un registro de datos de informe a partir del manejador de informes.

**Conceptos relacionados** Visión general de informes de EGL  
Visión general del proceso de creación de informes de EGL

**Tareas relacionadas**

Crear un manejador de informes de EGL  
Crear manualmente un manejador de informes de EGL  
Escribir código para controlar un informe

**Consulta relacionada**

Biblioteca de informes de EGL  
Tipos de datos en documentos de diseño XML

---

## Utilizar plantillas de informe

Puede seleccionar y modificar cualquiera de las plantillas de informes de EGL siguientes:

- Plantilla de conexión de base de datos
- Plantilla de datos de informe
- Plantilla de sentencia de SQL
- Plantilla de manejador de informes

Para utilizar una plantilla de informes, siga estos pasos:

1. Seleccione **Ventana > Preferencias**.
2. Cuando se muestre una lista de preferencias, expanda **EGL**.
3. Expanda **Editor** y seleccione **Plantillas**.
4. Desplácese por la lista de plantillas y seleccione una plantilla. Por ejemplo, seleccione **handler** para visualizar la plantilla de manejador de informes.
5. Pulse **Editar**.
6. Cambie la plantilla para que se ajuste a sus necesidades.

Teclee **handler** seguido de **Ctrl+espacio** para editar la plantilla de manejador de informes. Para obtener más información, incluyendo los ejemplos de código, consulte el apartado Crear manualmente un manejador de informes de EGL.

Teclee **jas** seguido de **Ctrl+espacio** para editar una plantilla de orígenes de datos.

7. Pulse **Aplicar** y después **Aceptar** para guardar los cambios.

**Conceptos relacionados**

Visión general de informes de EGL  
Visión general del proceso de creación de informes de EGL

**Tareas relacionadas**

Crear manualmente un manejador de informes de EGL  
Escribir código para controlar un informe  
Utilizar la ayuda de contenido en EGL  
Establecer las preferencias de plantillas

**Consulta relacionada**

Manejador de informes de EGL

## Crear un manejador de informes de EGL

Un manejador de informes de EGL proporciona la lógica para manejar eventos que se producen cuando se rellena un informe. Para crear un manejador de informes de EGL, haga lo siguiente:

1. Identifique un proyecto o carpeta para que contenga el archivo. Debe crear un proyecto o carpeta si no tiene uno todavía.
2. En el entorno de trabajo, pulse **Archivo > Nuevo > Otros**.
3. En la ventana Nuevo, expanda **EGL**.
4. Pulse **Manejador de informes**.
5. Pulse en **Siguiente**.
6. Seleccione el proyecto o carpeta que contendrá el archivo EGL y, a continuación, seleccione un paquete.
7. Dado que el nombre de manejador de informes será idéntico al nombre de archivo, elija un nombre de archivo que se ajuste a los convenios de denominación de componentes de EGL. En el campo Nombre de archivo fuente EGL, teclee el nombre del archivo EGL, por ejemplo myReportHandler.
8. Pulse en **Finalizar**.

### Conceptos relacionados

“Proceso de desarrollo” en la página 8

“Proyectos, paquetes y archivos EGL” en la página 13

“Salida generada” en la página 529

“Componentes” en la página 17

“Configuraciones de tiempo de ejecución” en la página 9

### Tareas relacionadas

“Crear un proyecto Web EGL” en la página 126

### Consulta relacionada

“Editor EGL” en la página 483

“Formato fuente EGL” en la página 491

---

## Crear manualmente un manejador de informes de EGL

Si no desea utilizar el asistente Manejador de informes EGL nuevo para crear un manejador de informes, puede crear manualmente el manejador de informes.

Para crear manualmente el manejador de informes, siga estos pasos:

1. Cree un archivo fuente EGL nuevo.
2. Lleve a cabo una de las dos acciones siguientes:
  - Especifique manualmente el código del manejador.
  - Inserte un manejador utilizando la plantilla de manejador de informes de la manera siguiente:
    - a. Vaya a la plantilla del manejador de informes y seleccione la plantilla que desea.
    - b. Pulse **Editar**.
    - c. Teclee **handler** seguido de **Ctrl+espacio**.
    - d. Cambie el código de la plantilla según sea necesario.

El resto de este tema contiene ejemplos de código que muestran lo siguiente:

- La sintaxis para crear manualmente un manejador de informes
- Cómo obtener parámetros de informe en un manejador de informes
- Cómo establecer y obtener variables de informe
- Cómo obtener valores de campo
- Cómo añadir un registro flexible
- Cómo un documento de diseño XML obtiene un registro de datos de informe del manejador de informes

Puede copiar este código y modificarlo para su aplicación.

### **El código de ejemplo que muestra la sintaxis para crear manualmente un manejador de informes**

El código siguiente muestra la sintaxis general para crear manualmente un manejador de informes de EGL:

```
handler handlerName type jasperReport

// Declaraciones de uso (opcional)
use usePartReference;

// Declaraciones de constante (opcional)
const constantName constantType = literal;

// Declaraciones de datos (opcional)
identifierName declarationType;

// Funciones de retorno de llamada Jasper predefinidas (opcional)
function beforeReportInit()
...
end

function afterReportInit()
...
end

function beforePageInit()
...
end

function afterPageInit()
...
end

function beforeColumnInit()
...
end

function afterColumnInit()
...
end

function beforeGroupInit(stringVariable string)
...
end

function afterGroupInit(stringVariable string)
...
end

function beforeDetailEval()
...
```

```

end

function afterDetailEval()
...
end

// Funciones definidas por el usuario (opcional)
function myFirstFunction()
...
end

function mySecondFunction()
...
end
end

```

### Ejemplo que muestra cómo obtener parámetros de informe

El fragmento de código siguiente muestra cómo obtener parámetros de informe en un manejador de informes:

```

handler myReportHandler type jasperReport

// Declaraciones de datos
report_title String;

// Función de retorno de llamada Jasper
function beforeReportInit()
...

 report_title = getReportTitle();

...
end

...

// Función definida por el usuario
function getReportTitle() Returns (String)
 return (getReportParameter("ReportTitle"));
end

end

```

### Ejemplo que muestra cómo establecer y obtener variables de informe

El fragmento de código siguiente muestra cómo establecer y obtener variables de informe en un manejador de informes:

```

handler myReportHandler type jasperReport

// Declaraciones de datos
employee_serial_number int;

// Función de retorno de llamada Jasper
function afterPageInit()
...
 employee_serial_number = getSerialNumberVar();
...
end

...

// Función definida por el usuario
function getSerialNumberVar() Returns (int)
 employeeName String;

```

```

 employeeName = "Ficus, Joe";
 setReportVariableValue("employeeNameVar", employeeName);
 return (getReportVariableValue("employeeSerialNumVar"));
 end
end

```

### Ejemplo que muestra cómo obtener valores de campo de informe en un manejador de informes

El fragmento de código de ejemplo siguiente muestra cómo obtener valores de campo de informe en un manejador de informes

```

handler myReportHandler type jasperReport

 // Declaraciones de datos
 employee_first_name String;

 // Función de retorno de llamada Jasper
 function beforeColumnInit()
 ...
 employee_first_name = getFirstNameField();
 ...
 end

 ...

 // Función definida por el usuario
 function getFirstNameField() Returns (String)
 fldName String;
 fldName = "fname";
 return (getFieldValue(fldName));
 end

end

```

### Ejemplo que muestra cómo añadir un registro flexible de datos de informe en un manejador de informes

El código de ejemplo siguiente muestra cómo añadir un registro flexible de datos de informe en un manejador de informes:

```

handler myReportHandler type jasperReport

 // Declaraciones de datos
 customer_array customerRecordType[];
 c customerRecordType;

 // Función de retorno de llamada Jasper
 function beforeReportInit()
 customer ReportData;
 datasetName String;

 //crear el objeto ReportData para el subinforme Customer
 c.customer_num = getFieldValue("c_customer_num");
 c.fname = getFieldValue("c_fname");
 c.lname = getFieldValue("c_lname");
 c.company = getFieldValue("c_company");
 c.address1 = getFieldValue("c_address1");
 c.address2 = getFieldValue("c_address2");
 c.city = getFieldValue("c_city");
 c.state = getFieldValue("c_state");
 c.zipcode = getFieldValue("c_zipcode");
 c.phone = getFieldValue("c_phone");
 customer_array.appendElement(c);
 customer.data = customer_array;
 end
end

```

```

 datasetName = "customer";
 addReportData(customer, datasetName);
 end
end

```

### Ejemplo que muestra cómo un documento de diseño XML obtiene un registro de datos de informe del manejador de informes

El fragmento de código siguiente muestra cómo un documento de diseño XML obtiene un registro flexible de datos de informe del manejador de informes:

```

<jasperReport name="MasterReport"
 scriptletClass="subreports.SubReportHandler">
 ...
 <subreport>
 <dataSourceExpression>
 <![CDATA[(JRDataSource)(((subreports.SubReportHandler)
 ${REPORT_SCRIPTLET}).getReportData(
 new String("customer")))]>
 </dataSourceExpression>
 <subreportExpression class="java.lang.String">
 <![CDATA["C:/RAD/workspaces/Customer.jasper"]>
 </subreportExpression>
 </subreport>
 ...
</jasperReport>

```

#### Conceptos relacionados

Visión general del informe EGL

Visión general del proceso de creación de informes de EGL

#### Tareas relacionadas

Crear un archivo fuente EGL

Crear un manejador de informes de EGL

Utilizar plantillas de informe

#### Consulta relacionada

Biblioteca de informes de EGL

Manejador de informes de EGL

Funciones de manejador de informes predefinidas

Funciones de manejador de informes EGL adicionales

---

## Escribir código para controlar un informe

Utilice el asistente Componente de programa EGL nuevo para crear un programa básico EGL nuevo que utiliza la biblioteca de informes para ejecutar los informes.

Para crear este controlador de informes, siga estos pasos:

1. Elija **Archivo > Nuevo > Programa** y después seleccione la carpeta que albergará el archivo EGL.
2. Seleccione un paquete.
3. Especifique un nombre de archivo para el archivo origen, seleccione el tipo *BasicProgram* y pulse **Finalizar**.
4. Busque la línea de programa.
5. En la función `main()`, justo después de la línea de programa, teclee **jas** seguido de **Ctrl+espacio** para insertar código para el controlador de informes.

6. En la ventana que contiene código y tipos de conexión de origen de datos, seleccione uno de los tipos de conexión de origen de datos.
7. Puede modificar el código existente o añadir su propio código. Si modifica el código, inserta valores específicos para las variables utilizadas por el controlador de informes. Estas variables son las siguientes:  
*reportDesignFileName, reportDestinationFileName, exportReportFile, alias, databaseName, userid, password y connectionName.*

### Código que muestra la información de invocación de informes

El código siguiente muestra la información de invocación de informes:

```
myReport Report;
myReportData ReportData;

myReport.reportDesignFile = "myReport_XML.jasper";
myReport.reportDestinationFile = "myReport.jrprint";
myReport.reportExportFile = "myReport.pdf";

myReportData.sqlStatement = "Select * From myTable";

myReport.reportData = myReportData;

ReportLib.fillReport(myReport, DataSource.sqlStatement);

ReportLib.exportReport(myReport, ExportFormat.pdf);
```

| Código                                                    | Descripción                                                                                |
|-----------------------------------------------------------|--------------------------------------------------------------------------------------------|
| myReport Report;                                          | Esto es una declaración de registro de biblioteca de informes                              |
| myReportData ReportData;                                  | Esto es una declaración de registro de datos de biblioteca de informes                     |
| myReport.reportDesignFile = "myReport_XML.jasper";        | Esta sentencia define el diseño de informe que se debe utilizar para crear un informe.     |
| myReport.reportDestinationFile = "myReport.jrprint";      | Esta sentencia especifica el nombre de archivo para la salida de informe generada.         |
| myReport.reportExportFile = "myReport.pdf"                | Esta sentencia especifica el nombre de archivo para la salida exportada.                   |
| myReport.sqlStatement = "Select * From myTable";          | Esto proporciona información acerca de la sentencia Select de SQL utilizada en el informe. |
| myReport.reportData = myReportData;                       | Esto proporciona información acerca de los datos del informe.                              |
| ReportLib.fillReport(myReport, DataSource.sqlStatement ); | Esta sentencia especifica información de origen para el informe.                           |
| ReportLib.exportReport(myReport, ExportFormat.pdf);       | Esta sentencia especifica el formato de salida del informe.                                |

### Fragmento de código de ejemplo:

```
//ubicación en la que se almacena el archivo .jrprint.
abcReport.reportDestinationFile="C:\\temp\\MasterReport.jrprint";

//ubicación para el informe exportado.
abcReport.reportExportFile="C:\\temp\\MasterReport.pdf";

//realizar la exportación.
ReportLib.exportReport(abcReport, ExportFormat.pdf);
```

### Conceptos relacionados

Visión general del informe EGL

Visión general del proceso de creación de informes de EGL

### Tareas relacionadas

Crear un manejador de informes de EGL

Crear manualmente un manejador de informes de EGL

Utilizar plantillas de informe

### Consulta relacionada

Manejador de informes de EGL

Biblioteca de informes de EGL

Código de ejemplo para funciones de controlador de informes de EGL

---

## Generar archivos para un informe y ejecutarlo

Debe tener un documento de diseño XML con una extensión `.jrxml` que especifique la información de diseño del informe.

Para construir y generar un informe para un proyecto EGL, siga estos pasos:

1. Construya el proyecto EGL seleccionando **Proyecto > Construir todo**.  
EGL genera automáticamente código Java a partir del manejador de informes EGL y compila el documento de diseño XML (el archivo `.jrxml`) en un archivo `.jasper`.
2. Ejecute el programa EGL que tiene el código de invocación de informe. Para hacer esto en el Explorador de paquetes, pulse con el botón derecho del ratón sobre el archivo `.egl` que contiene el código. A continuación, seleccione **Generar** en el menú emergente.

Además de generar el informe, el programa JasperReports utilizado por EGL guarda automáticamente el informe generado en la ubicación especificada por `reportDestinationFileName` en el código de invocación de informes.

El programa JasperReports que genera el informe también genera y almacena un archivo `.jprint` que es un formato de archivo intermedio que se exporta al formato de informe final (`.pdf`, `.html`, `.xml`, `.txt` o `.csv`). El programa puede reutilizar un archivo `.jprint` para varias exportaciones.

La función `exportReport()` del código de invocación de informes hace que EGL exporte el informe en el formato especificado.

**Conceptos relacionados** Visión general de informes de EGL  
Visión general del proceso de creación de informes de EGL

### Tareas relacionadas

Crear un manejador de informes de EGL

Crear manualmente un manejador de informes de EGL

Escribir código para controlar un informe

Exportar informes

### Consulta relacionada

Biblioteca de informes de EGL

Manejador de informes de EGL

---

## Exportar informes

Puede exportar informes cumplimentados como PDF, HTML, XML, CSV (valores separados por comas) y salida de texto plano. La función *exportReport()* del código de controlador de informes de EGL insta a EGL a exportar el informe en el formato especificado.

El valor *exportReportFile* del código de controlador de informes especifica la ubicación del archivo exportado.

Para especificar el formato de los informes exportados, utilice uno de los parámetros siguientes de la llamada a la función *exportReport()*:

- ExportFormat.html
- ExportFormat.pdf
- ExportFormat.text
- ExportFormat.xml
- ExportFormat.csv

Por ejemplo, el código siguientes hace que EGL exporte un informe en formato .pdf:

```
reportLib.exportReport(myReport, ExportFormat.pdf);
```

**Importante:** EGL no renueva automáticamente informes exportados. Si cambia el diseño del informe o si los datos cambian, deberá volver a confeccionar y a exportar el informe.

**Conceptos relacionados** Visión general de informes de EGL  
Visión general del proceso de creación de informes de EGL

### Tareas relacionadas

Crear un manejador de informes de EGL  
Crear manualmente un manejador de informes de EGL  
Escribir código para controlar un informe  
Ejecutar un informe

### Consulta relacionada

Biblioteca de informes de EGL  
Manejador de informes de EGL

---

## Trabajar con archivos y bases de datos

---

### Soporte de SQL

Como se muestra en la tabla siguiente, el código generado por EGL puede acceder a una base de datos relacional en cualquiera de los sistemas destino.

| Sistema destino                                                     | Soporte para el acceso a bases de datos relacionales |
|---------------------------------------------------------------------|------------------------------------------------------|
| Servicios del sistema AIX, iSeries, Linux, Windows 2000/NT/XP, UNIX | JDBC proporciona acceso a DB2 UDB, Oracle o Informix |

Mientras trabaja en un programa, puede codificar las sentencias SQL de la misma manera que lo haría para codificar programas en la mayoría de los demás lenguajes. Para facilitarle el trabajo, EGL proporciona plantillas de sentencias SQL para que las rellene.

Como alternativa, puede utilizar un registro SQL como objeto de E/S cuando codifique una sentencia EGL. Esta utilización del registro significa que accede a una base de datos personalizando una sentencia SQL que se le ha proporcionado o bien dependiendo de un valor por omisión que elimina la necesidad de codificar SQL.

En cualquiera de los dos casos, tenga en cuenta los siguientes aspectos del soporte de EGL:

- Si desea probar un nulo en una determinada columna de tabla, debe recibir el valor de columna en un registro SQL, en un elemento de registro que esté declarado con posibilidad de nulos. Para obtener información detallada, consulte la sección *Probar y establecer NULL* que se describe más adelante.
- En las secciones de visión general siguientes (y de acuerdo con la terminología SQL), cada elemento al que se hace referencia en una sentencia SQL recibe el nombre de *variable del lenguaje principal*. La palabra *host* hace referencia al lenguaje que se incorpora en la sentencia SQL; en este caso, al lenguaje de procedimiento EGL. Una variable del lenguaje principal en una sentencia SQL va precedido por un carácter de dos puntos, como en el siguiente ejemplo:

```
select empnum, empname
from employee
where empnum >= :myRecord.empnum
for update of empname
```

### SQL y sentencias EGL

La tabla siguiente muestra las palabras clave EGL que puede utilizar para acceder a una base de datos relacional. Esta tabla también incluye una descripción de las sentencias SQL correspondientes a cada palabra clave. Cuando se codifica una sentencia EGL **add**, por ejemplo, se genera una sentencia SQL INSERT.

En muchas aplicaciones empresariales, se utiliza la sentencia **open** de EGL y diversas clases de sentencias **get by position**. El código ayuda a declarar, abrir y procesar un *cursor*, que es una entidad de tiempo de ejecución que actúa de la forma siguiente:

- Devuelve un *conjunto de resultados*, que es una lista de filas que cumplen los criterios de búsqueda
- Señala a una fila específica del conjunto de resultados

Puede utilizar la sentencia EGL **open** para llamar a un procedimiento almacenado. Este procedimiento se compone de lógica que está escrita fuera de EGL, se almacena en el sistema de gestión de bases de datos y también devuelve un conjunto de resultados. (Además, puede utilizar la sentencia EGL **execute** para llamar a un procedimiento almacenado).

En secciones siguientes se proporcionan detalles sobre cómo procesar un conjunto de resultados.

Si tiene la intención de codificar explícitamente sentencias SQL, utilice la sentencia EGL **execute** y posiblemente la sentencia EGL **prepare**.

| Palabra clave/finalidad                                                                                                                                                                                     | Descripción de las sentencias SQL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | ¿Se puede modificar la SQL? |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------|
| add<br><br>Coloca una fila en una base de datos; o bien (si se utiliza una matriz dinámica de registros SQL), coloca un conjunto de filas en función del contenido de los elementos sucesivos de la matriz. | INSERT row (ya que aparece repetidamente, si especifica una matriz dinámica).                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Sí                          |
| close<br><br>Libera las filas no procesadas.                                                                                                                                                                | CLOSE cursor.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | No                          |
| delete<br><br>Suprime una fila de una base de datos.                                                                                                                                                        | DELETE row. La fila se ha seleccionado de una de las dos maneras siguientes: <ul style="list-style-type: none"> <li>• Al invocar una sentencia <b>get</b> con la opción forUpdate (según sea apropiado cuando desee seleccionar la primera de varias filas que tienen el mismo valor de clave)</li> <li>• Al invocar una sentencia <b>open</b> con la opción forUpdate y luego una sentencia <b>get next</b> (según sea apropiado cuando desee seleccionar un conjunto de filas y procesar los datos recuperados en un bucle)</li> </ul> | No                          |

| Palabra clave/finalidad                                                                                                                                                                                                                                                                                             | Descripción de las sentencias SQL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | ¿Se puede modificar la SQL? |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------|
| <p>forEach</p> <p>Marca el inicio de un conjunto de sentencias que se ejecutan en un bucle. La primera iteración se produce solamente si un conjunto de resultados especificado está disponible y continúa (en la mayoría de los casos) hasta que se ha procesado la última fila de ese conjunto de resultados.</p> | <p>EGL convierte una sentencia <b>forEach</b> en una sentencia FETCH de SQL que se ejecuta en un bucle.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | <p>No</p>                   |
| <p>freeSQL</p> <p>Libera los recursos asociados a una sentencia SQL preparada dinámicamente, cerrando cualquier cursor abierto asociado con esa sentencia SQL.</p>                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | <p>No</p>                   |
| <p>get (también llamada <b>get by key value</b>)</p> <p>Lee una única fila de una base de datos; o bien (si utiliza una matriz dinámica de registros SQL), lee filas sucesivas de elementos sucesivos de la matriz.</p>                                                                                             | <p>SELECT row, pero sólo si establece la opción singleRow. En caso contrario, se aplican las siguientes normas:</p> <ul style="list-style-type: none"> <li>• EGL convierte una sentencia <b>get</b> en: <ul style="list-style-type: none"> <li>– DECLARE cursor con SELECT o (si establece la opción forUpdate) con SELECT FOR UPDATE.</li> <li>– OPEN cursor.</li> <li>– FETCH row.</li> </ul> </li> <li>• Si no ha especificado la opción forUpdate, EGL también cierra el cursor.</li> <li>• Las opciones singleRow y forUpdate no están soportadas con matrices dinámicas; en este caso, el entorno de ejecución EGL declara y abre un cursor, obtiene una serie de filas y cierra el cursor.</li> </ul> | <p>Sí</p>                   |
| <p>get absolute</p> <p>Lee una fila especificada numéricamente en un conjunto de resultados seleccionado por una sentencia <b>open</b>.</p>                                                                                                                                                                         | <p>EGL convierte una sentencia <b>get absolute</b> en una sentencia FETCH de SQL.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | <p>No</p>                   |
| <p>get current</p> <p>Lee la flecha en la que el cursor ya está posicionado en un conjunto de resultados seleccionado por una sentencia <b>open</b>.</p>                                                                                                                                                            | <p>EGL convierte una sentencia <b>get current</b> en una sentencia FETCH de SQL.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | <p>No</p>                   |

| Palabra clave/finalidad                                                                                                                                                                                                                                                     | Descripción de las sentencias SQL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | ¿Se puede modificar la SQL? |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------|
| <p>get first</p> <p>Lee la primera fila de un conjunto de resultados seleccionado por una sentencia <b>open</b>.</p>                                                                                                                                                        | <p>EGL convierte una sentencia <b>get first</b> en una sentencia FETCH de SQL.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | <p>No</p>                   |
| <p>get last</p> <p>Lee la última fila de un conjunto de resultados seleccionado por una sentencia <b>open</b>.</p>                                                                                                                                                          | <p>EGL convierte una sentencia <b>get last</b> en una sentencia FETCH de SQL.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | <p>No</p>                   |
| <p>get next</p> <p>Lee la fila siguiente de un conjunto de resultados seleccionado por una sentencia <b>open</b>.</p>                                                                                                                                                       | <p>EGL convierte una sentencia <b>get next</b> en una sentencia SQL FETCH.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | <p>No</p>                   |
| <p>get previous</p> <p>Lee la fila anterior de un conjunto de resultados seleccionado por una sentencia <b>open</b>.</p>                                                                                                                                                    | <p>EGL convierte una sentencia <b>get previous</b> en una sentencia FETCH de SQL.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | <p>No</p>                   |
| <p>get relative</p> <p>Lee una fila especificada numéricamente en un conjunto de resultados seleccionado por una sentencia <b>open</b>. La fila se identifica en relación con la posición del cursor en el conjunto de resultados.</p>                                      | <p>EGL convierte una sentencia <b>get relative</b> en una sentencia FETCH de SQL.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | <p>No</p>                   |
| <p>execute</p> <p>Permite ejecutar una sentencia de definición de datos SQL (de tipo CREATE TABLE, por ejemplo); o una sentencia de manipulación de datos (de tipo INSERT o UPDATE, por ejemplo); o una sentencia SQL preparada que no empieza con una cláusula SELECT.</p> | <p>La sentencia SQL que se escribe está disponible en el sistema de gestión de bases de datos.</p> <p>El uso principal de <b>execute</b> es codificar una única sentencia SQL que se formatea completamente durante la generación, como en el siguiente ejemplo:</p> <pre>try   execute     #sql{ // sin ningún espacio después de "#sql"       delete         from EMPLOYEE         where department =           :myRecord.department     }; onException   myErrorHandler(10); end</pre> <p>Una sentencia SQL completamente formateada puede incluir variables del lenguaje principal en la cláusula WHERE.</p> | <p>Sí</p>                   |

| Palabra clave/finalidad                                                                                                                                                                                                                                                                              | Descripción de las sentencias SQL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | ¿Se puede modificar la SQL? |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------|
| <p>open</p> <p>Selecciona un conjunto de filas a partir de una base de datos relacional para recuperarlas posteriormente con las sentencias <b>get next</b>.</p>                                                                                                                                     | <p>EGL convierte una sentencia <b>open</b> en una sentencia CALL (para acceder a un procedimiento almacenado) o en las siguientes sentencias:</p> <ul style="list-style-type: none"> <li>• DECLARE cursor con SELECT o con SELECT FOR UPDATE.</li> <li>• OPEN cursor.</li> </ul>                                                                                                                                                                                                                                                                                                                                                         | Sí                          |
| <p>prepare</p> <p>Especifica una sentencia SQL PREPARE, que opcionalmente incluye detalles que sólo se conocen durante la ejecución; ejecute la sentencia SQL preparada ejecutando una sentencia EGL execute o (si la sentencia SQL empieza con SELECT) ejecutando una sentencia EGL open o get.</p> | <p>EGL convierte una sentencia <b>prepare</b> en una sentencia SQL PREPARE, que siempre se construye durante la ejecución. En el siguiente ejemplo de una sentencia EGL <b>prepare</b>, cada marcador de parámetro (?) se resuelve mediante la cláusula USING en la sentencia <b>execute</b> subsiguiente:</p> <pre>myString =   "insert into myTable " +   "(empnum, empname) " +   "value ?, ?";  try   prepare myStatement   from myString; onException   // salir del programa   myErrorHandler(12); end  try   execute myStatement   using :myRecord.empnum,         :myRecord.empname; onException   myErrorHandler(15); end</pre> | Sí                          |
| <p>replace</p> <p>Vuelve a poner una fila cambiada en una base de datos.</p>                                                                                                                                                                                                                         | <p>UPDATE row. La fila se ha seleccionado de una de las dos maneras siguientes:</p> <ul style="list-style-type: none"> <li>• Al invocar una sentencia <b>get</b> con la opción forUpdate (según sea apropiado cuando desee seleccionar la primera de varias filas que tienen el mismo valor de clave); o bien</li> <li>• Al invocar una sentencia <b>open</b> con la opción forUpdate y luego una sentencia <b>get next</b> (según sea apropiado cuando desee seleccionar un conjunto de filas y procesar los datos recuperados en un bucle).</li> </ul>                                                                                 | Sí                          |

**Nota:** En ningún caso se pueden actualizar varias tablas de base de datos codificando una única sentencia EGL.

## Proceso del conjunto de resultados

Una forma habitual de actualizar una serie de filas es la siguiente:

1. Declare y abra un cursor ejecutando una sentencia EGL **open** con la opción forUpdate; esta opción hace que las filas seleccionadas se bloqueen para una posterior actualización o supresión

2. Obtenga una fila ejecutando una sentencia EGL **get next**
3. Realice lo siguiente en un bucle:
  - a. Cambie los datos de las variables del lenguaje principal en las que ha recuperado los datos
  - b. Actualice la fila ejecutando una sentencia EGL **replace**
  - c. Obtenga otra fila ejecutando la sentencia EGL **get next**
4. Comprometa los cambios ejecutando la función EGL **commit**.

Las sentencias que abren el cursor y que actúan sobre las filas de dicho cursor están relacionadas entre sí mediante un identificador de conjunto de resultados, que debe ser exclusivo entre todos los identificadores de conjunto de resultados, variables de programa y parámetros de programa dentro del programa. Especifique dicho identificador en la sentencia **open** que abre el cursor, y haga referencia al mismo identificador en las sentencias **get next**, **delete** y **replace** que afectan a una fila individual, así como en la sentencia **close** que cierra el cursor. Para obtener información detallada, consulte la sección *resultSetID*.

El código siguiente muestra cómo actualizar una serie de filas cuando codifique SQL usted mismo:

```

VGVar.handleHardIOErrors = 1;

try
 open selectEmp forUpdate with
 #sql{ // sin ningún espacio después de "#sql"
 select empname
 from EMPLOYEE
 where empnum >= :myRecord.empnum
 for update of empname
 };

onException
 myErrorHandler(8); // sale del programa
end

try
 get next from selectEmp into :myRecord.empname;
onException
 if (sysVar.sqlcode != 100)
 myErrorHandler(8); // salir del programa
 end
end

while (sysVar.sqlcode != 100)
 myRecord.empname = myRecord.empname + " " + "III";

 try
 execute
 #sql{
 update EMPLOYEE
 set empname = :empname
 where current of selectEmp
 };
 onException
 myErrorHandler(10); // sale del programa
 end

 try
 get next from selectEmp into :myRecord.empname;
 onException
 if (sysVar.sqlcode != 100)
 myErrorHandler(8); // sale del programa
 end
 end
end

```

```

end
end // end while; el cursor se cierra automáticamente
// cuando se lee la última fila del conjunto de resultados

sysLib.commit;

```

Si desea evitar parte de la complejidad del ejemplo anterior, considere la posibilidad de utilizar registros SQL. Estos registros permiten perfeccionar el código y utilizar valores de error de E/S que no varían en los diferentes sistemas de gestión de bases de datos. El siguiente ejemplo es equivalente al anterior pero utiliza un registro SQL llamado emp:

```

VGVar.handleHardIOErrors = 1;

try
 open selectEmp forUpdate for emp;
onException
 myErrorHandler(8); // sale del programa
end

try
 get next emp;
onException
 if (sysVar.sqlcode not noRecordFound)
 myErrorHandler(8); // salir del programa
 end
end

while (sysVar.sqlcode not noRecordFound)
 myRecord.empname = myRecord.empname + " " + "III";

 try
 replace emp;
onException
 myErrorHandler(10); // sale del programa
end

 try
 get next emp;
on exception
 if (sysVar.sqlcode not noRecordFound)
 myErrorHandler(8); // sale del programa
 end
 end
end // end while; el cursor se cierra automáticamente
// cuando se lee la última fila del conjunto de resultados

sysLib.commit;

```

En las secciones siguientes se describen los registros SQL.

## Registros SQL y sus usos

Un registro SQL es una variable que se basa en un componente de registro SQL. Este tipo de registro permite interactuar con una base de datos relacional como si se estuviera accediendo a un archivo. Si, por ejemplo, la variable EMP se basa en un componente de registro SQL que hace referencia a la tabla de base de datos EMPLOYEE, se puede utilizar EMP en una sentencia EGL **add**:

```
add EMP;
```

En este caso, EGL inserta los datos de EMP en EMPLOYEE. El registro SQL también incluye información de estado de modo que después de ejecutarse la

sentencia EGL, se puede probar el registro SQL para que realice tareas condicionalmente, de acuerdo con el valor de error de E/S que se ha obtenido del acceso a la base de datos:

```
VGVar.handleHardIOErrors = 1;

try
 add EMP;
onException
 if (EMP is unique) // si una fila de la tabla
 // tenía la misma clave
 myErrorHandler(8);
 end
end
```

Un registro SQL, como por ejemplo EMP, permite interactuar con una base de datos relacional del modo siguiente:

- Declare un componente de registro SQL y el registro SQL relacionado
- Defina un conjunto de sentencias EGL de modo que cada una de ellas utilice el registro SQL como un objeto de E/S
- Acepte el comportamiento por omisión de las sentencias EGL o bien realice los cambios en SQL que sean apropiados para la lógica empresarial

### **Declarar un componente de registro SQL y el registro relacionado**

Declare un componente de registro SQL y asocie cada uno de los elementos de registro con una columna de la tabla o vista relacional. Puede permitir que EGL realice esta asociación automáticamente utilizando la característica de recuperación del editor EGL, como se describe más adelante en la sección *Acceso a base de datos durante la declaración*.

Si el componente de registro SQL no es un componente de registro fijo, puede incluir campos primitivos así como otras variables. Probablemente incluirá las clases de variables siguientes:

- Otros registros SQL. La presencia de cada uno representa una relación uno a uno entre las tablas padre e hijo.
- Matrices de registros SQL. La presencia de cada uno representa una relación de uno a muchos entre las tablas padre e hijo.

Solo los campos de un tipo primitivo pueden representar una columna de base de datos.

si los números de nivel preceden a los campos, el componente de registro SQL es un componente de registro fijo. Se aplican las siguientes normas:

- La estructura de cada componente de registro SQL debe ser *plana* (sin jerarquía).
- Todos los campos deben ser campos primitivos, pero no de tipo BLOB, CLOB o STRING
- Ninguno de los campos de registro puede ser una matriz de campo de estructura

Después de declarar un componente de registro SQL, declare un registro SQL que se base en dicho componente.

### **Definir las sentencias EGL relacionadas con SQL**

Puede definir un conjunto de sentencias EGL de modo que cada una de ellas utilice el registro SQL como objeto de E/S de la sentencia. Para cada sentencia,

EGL proporciona una *sentencia SQL implícita*, que no está en el fuente pero está implícita por la combinación del registro SQL y la sentencia EGL. Por ejemplo, en el caso de una sentencia EGL **add**, una sentencia SQL INSERT implícita coloca el valor de un determinado elemento de registro en la columna de tabla asociada. Si el registro SQL incluye un elemento de registro para el que no se ha asignado ninguna columna de tabla, EGL forma la sentencia SQL implícita suponiendo que el nombre del elemento de registro es idéntico al nombre de la columna.

**Utilizar sentencias SELECT implícitas:** Cuando se define una sentencia EGL que utiliza un registro SQL y que genera una sentencia SQL SELECT o una declaración de cursor, EGL proporciona una sentencia SQL SELECT implícita. (Esta sentencia se incorpora en la declaración de cursor, si existe). Por ejemplo, podría declarar una variable llamada EMP que estuviera basada en el siguiente componente de registro:

```
Record Employee type sqlRecord
{ tableNames = [{"EMPLOYEE"}],
 keyItems = ["empnum"] }
empnum decimal(6,0);
empname char(40);
end
```

A continuación, podría codificar una sentencia **get**:

```
get EMP;
```

La sentencia SQL SELECT implícita es la siguiente:

```
SELECT empnum, empname
FROM EMPLOYEE
WHERE empnum = :empnum
```

EGL también coloca una cláusula INTO en la sentencia SELECT autónoma (si no hay ninguna declaración de cursor) o en la sentencia FETCH asociada al cursor. La cláusula INTO lista las variables del lenguaje principal que reciben valores de las columnas listadas en la primera cláusula de la sentencia SELECT:

```
INTO :empnum, :empname
```

La sentencia SELECT implícita lee el valor de cada columna en la variable del lenguaje principal correspondiente; hace referencia a las tablas especificadas en el registro SQL; y tiene un criterio de búsqueda (una cláusula WHERE) que depende de *una combinación de dos factores*:

- El valor que se ha especificado para la propiedad de registro **defaultSelectCondition**; y
- Una relación (como por ejemplo una igualdad) entre dos conjuntos de valores:
  - Los nombres de las columnas que constituyen la clave de tabla
  - Los valores de las variables del lenguaje principal que constituyen la clave de registro

Se produce una situación especial si lee datos en una matriz dinámica de registros SQL, como ocurre con la sentencia **get**:

- Un cursor está abierto, filas sucesivas de la base de datos se leen en elementos sucesivos de la matriz, el conjunto de resultados se libera y el cursor se cierra.
- Si no se especifica una sentencia SQL, el criterio de búsqueda depende de la propiedad de registro **defaultSelectCondition**, pero también depende de una relación (concretamente, una relación de mayor o igual que) entre los siguientes conjuntos de valores:

- Los nombres de columnas, como se especifica indirectamente al especificar elementos en la sentencia EGL
- Los valores de dichos elementos

Todas las variables del lenguaje principal especificadas en la propiedad **defaultSelectCondition** deben estar fuera del registro SQL que es la base de la matriz dinámica.

Para obtener información detallada sobre la sentencia SELECT implícita, que varía según la palabra clave, consulte las secciones *get* y *open*.

**Utilizar registros SQL con cursores:** Cuando utilice registros SQL, puede relacionar las sentencias de proceso de cursores utilizando el mismo registro SQL en varias sentencias EGL, de la misma manera que puede hacerlo utilizando un identificador de conjunto de resultados. Sin embargo, cualquier relación entre sentencias que se indique mediante un identificador de conjunto de resultados tiene preferencia sobre una relación indicada mediante el registro SQL; y en algunos casos debe especificar un resultSetID.

Además, sólo un cursor puede estar abierto para un determinado registro SQL. Si una sentencia EGL abre un cursor cuando otro cursor está abierto para el mismo registro SQL, el código generado cierra automáticamente el primer cursor.

### Personalizar las sentencias SQL

Dada una sentencia EGL que utiliza un registro SQL como objeto de E/S, puede actuar de una de las dos maneras siguientes:

- Puede aceptar la sentencia SQL implícita. En este caso, los cambios realizados en el componente de registro SQL afectan a las sentencias SQL utilizadas durante la ejecución. Si, por ejemplo, más adelante indica que debe utilizarse un elemento de registro distinto como clave del registro SQL, EGL cambiará la sentencia SELECT implícita utilizada en cualquier declaración de cursor que se base en dicho componente de registro SQL.
- En su lugar, puede elegir hacer explícita la sentencia SQL. En este caso, los detalles de dicha sentencia SQL se aíslan del componente de registro SQL y los cambios subsiguientes que se realicen en el componente de registro SQL no tendrán ningún efecto sobre la sentencia SQL que se utilice durante la ejecución. Si se elimina una sentencia SQL explícita del fuente, la sentencia SQL implícita (si existe) vuelve a estar disponible durante la generación.

### Ejemplo de utilización de un registro en un registro

Para permitir que un programa recupere datos para una serie de empleados en un departamento, puede crear dos componentes de registro y una función, de la forma siguientes:

```
DataItem DeptNo { column = deptNo } end

Record Dept type SQLRecord
 deptNo DeptNo;
 managerID CHAR(6);
 employees Employee[];
end

Record Employee type SQLRecord
 employeeID CHAR(6);
 empDeptNo DeptNo;
end
```

```
Function getDeptEmployees(dept Dept)
 get dept.employees usingKeys dept.deptNo;
end
```

## Probar y establecer NULL

En algunos casos, EGL mantiene internamente un indicador nulo para un subconjunto de variables en el código. Si acepta el comportamiento por omisión, EGL mantiene internamente un indicador nulo para cada variable que tenga estas características:

- Está en un registro SQL
- Se declara con la propiedad **isNullable** establecida en *yes*

No codifique variables del lenguaje principal para indicadores de nulos en las sentencias SQL, como podría hacer en algunos lenguajes. Para probar los nulos de una variable del lenguaje principal con posibilidad de nulos, utilice una sentencia EGL **if**. También puede probar la recuperación de un valor truncado, pero sólo cuando un indicador de nulos está disponible.

Puede anular una columna de tabla SQL de una de las dos maneras siguientes:

- Utilice una sentencia EGL **set** para anular una variable del lenguaje principal con posibilidad de nulos y, a continuación, escriba el registro SQL relacionado en la base de datos; o bien
- Utilice la sintaxis SQL apropiada, escribiendo una sentencia SQL desde cero o personalizando una sentencia SQL que está asociada a la sentencia EGL **add** o **replace**

Para conocer más detalles sobre el proceso nulo, consulte las secciones *itemsNullable* y *Propiedades de elementos SQL*.

## Acceso a base de datos durante la declaración

El acceso (durante la declaración) a una base de datos que tiene características similares a la base de datos a la que accederá el código durante la ejecución tiene las siguientes ventajas:

- Si accede a una tabla o vista de base de datos que es equivalente a una tabla o vista asociada a un registro SQL, puede utilizar la característica de recuperación del editor de componentes EGL para crear o sobrescribir los elementos de registro. La característica de recuperación accede a la información almacenada en el sistema de gestión de bases de datos de modo que el número y las características de datos de los elementos creados refleja el número y las características de las columnas de tabla. Después de invocar la característica de recuperación, puede red denominar los elementos de registro, suprimir los elementos de registro y realizar otros cambios en el registro SQL.
- El acceso a una base de datos estructurada apropiadamente durante la declaración ayuda a asegurar que las sentencias SQL serán válidas en relación con una base de datos equivalente durante la ejecución.

La característica de recuperación crea elementos de registro cada uno de los cuales tiene el mismo nombre (o casi el mismo nombre) que la columna de tabla relacionada.

No se puede recuperar una vista que está definida con la condición de DB2 WITH CHECK OPTIONS.

Para obtener más información sobre cómo utilizar la característica de recuperación, consulte el apartado *Recuperar datos de tabla SQL*. Para obtener información detallada sobre denominación, consulte la sección *Establecer las preferencias de la recuperación SQL*.

Para acceder a una base de datos durante la declaración, especifique información de conexión en una página de preferencias, como se describe en la sección *Establecer las preferencias de las conexiones de base de datos SQL*.

### Conceptos relacionados

“SQL dinámico”

“Unidad lógica de trabajo” en la página 307

“resultSetID” en la página 743

### Tareas relacionadas

“Recuperar datos de tabla SQL” en la página 252

“Establecer preferencias para conexiones a bases de datos SQL” en la página 119

“Establecer preferencias para la recuperación de SQL” en la página 121

### Consulta relacionada

“add” en la página 561

“close” en la página 568

“Autorización de base de datos y nombres de tabla” en la página 466

“Base de datos por omisión” en la página 251

“delete” en la página 571

“execute” en la página 574

“get” en la página 585

“get next” en la página 597

“Informix y EGL” en la página 252

“itemsNullable” en la página 395

“open” en la página 616

“prepare” en la página 630

“replace” en la página 632

“Códigos de datos SQL y variables de lenguaje principal EGL” en la página 744

“Ejemplos de SQL” en la página 241

“Propiedades de elementos SQL” en la página 67

“Diseño interno de los registros SQL” en la página 747

“Componente de registro SQL en formato fuente EGL” en la página 748

“Probar y establecer NULL” en la página 239

## SQL dinámico

La sentencia SQL asociada a una sentencia EGL puede especificarse estáticamente, con todos los detalles establecidos durante la generación. Sin embargo, cuando está en vigor SQL dinámico, la sentencia SQL se construye durante la ejecución cada vez que se invoca la sentencia EGL.

La utilización de SQL dinámico disminuye la velocidad del proceso de ejecución, pero permite variar una operación de base de datos como respuesta a un valor de ejecución:

- Para una consulta de base de datos, es posible que desee variar los criterios de selección, la forma en que se agregan los datos o el orden en que se devuelven las filas; estos detalles se controlan mediante las cláusulas WHERE, HAVING, GROUP BY y ORDER BY. En este caso, puede utilizar la sentencia prepare.

- Para muchos tipos de operaciones, es posible que desee que un valor de ejecución determine a qué tabla se debe acceder. Puede llevar a cabo la especificación dinámica de una tabla de una de las dos maneras siguientes:
  - Utilice la sentencia `prepare`; o bien
  - Utilice un registro SQL y especifique un valor para la propiedad `tableNameVariables`, como se describe en *Componente de registro SQL en formato fuente EGL*.

### Conceptos relacionados

“Soporte de SQL” en la página 229

### Consulta relacionada

“Autorización de base de datos y nombres de tabla” en la página 466

“prepare” en la página 630

“Componente de registro SQL en formato fuente EGL” en la página 748

## Ejemplos de SQL

Puede acceder a una base de datos SQL de cualquiera de estas formas:

- Codificando manualmente una sentencia SQL cuyo formato se conozca durante la generación.
- Utilizando un registro SQL como objeto de E/S de una sentencia EGL, cuando el formato de la sentencia SQL se conoce durante la generación:
  - Si coloca una sentencia SQL explícita en el código fuente EGL, esa sentencia SQL se utiliza durante la ejecución;
  - De lo contrario, se utiliza una sentencia SQL implícita durante la ejecución.
- Codificando una sentencia EGL **prepare**, que genera una sentencia SQL `PREPARE` que, a su vez, crea una sentencia SQL durante la ejecución.

En cualquier caso, puede utilizar un registro SQL como área de memoria y para proporcionar una forma sencilla de comprobar si la operación ha sido satisfactoria. Los ejemplos de esta sección presuponen que se ha declarado un componente de registro en un archivo EGL y que se ha declarado un registro basado en el componente en un programa de ese archivo:

- El componente de registro SQL es el siguiente:

```
Record Employee type sqlRecord
{
 tableName = ["employee"],
 keyItems = ["empnum"],
 defaultSelectCondition =
 #sqlCondition{ // no deje espacios entre #sql y el corchete
 aTableColumn = 4 -- iniciar cada componente SQL
 -- con un guión doble
 }
}

empnum decimal(6,0) {isReadOnly=yes};
empname char(40);
end
```

- El registro SQL es el siguiente:

```
emp Employee;
```

Para obtener más detalles acerca de los registros y las sentencias implícitas SQL, consulte el apartado Soporte SQL.

## Codificar sentencias SQL

Para preparar la codificación de sentencias SQL, debe declarar variables:

```
empnum decimal(6,0);
empname char(40);
```

**Añadir una fila a una tabla SQL:** Para preparar la adición de una fila, asigne valores a las variables:

```
empnum = 1;
empname = "John";
```

Para añadir la fila, asocie una sentencia EGL **execute** con una sentencia SQL INSERT, del siguiente modo:

```
try
 execute
 #sql{
 insert into employee (empnum, empname)
 values (:empnum, :empname)
 };
onException
 myErrorHandler(8);
end
```

**Leer un conjunto de filas de una tabla SQL:** Para preparar la lectura de un conjunto de filas de una tabla SQL, identifique una clave de registro:

```
empnum = 1;
```

Para obtener los datos, codifique una serie de sentencias EGL:

- Para seleccionar un conjunto de resultados, ejecute una sentencia EGL open:

```
open selectEmp
 with #sql{
 select empnum, empname
 from employee
 where empnum >= :empnum
 for update of empname
 }
 into empnum, empname;
```

- Para acceder a la fila siguiente del conjunto de resultados, ejecute una sentencia EGL get next:

```
get next from selectEmp;
```

Si no ha especificado la cláusula into en la sentencia open, debe especificar la cláusula into en la sentencia get next; si ha especificado la cláusula into en ambas ubicaciones, la cláusula de la sentencia get next tiene preferencia:

```
get next from selectEmp
 into empnum, empname;
```

El cursor se cierra automáticamente cuando se lee el último registro del conjunto de resultados.

A continuación se proporciona un ejemplo de código más completo que actualiza un conjunto de filas:

```
VGVar.handleHardIOErrors = 1;

try
 open selectEmp
 with #sql{
 select empnum, empname
 from employee
 where empnum >= :empnum
 for update of empname
```

```

 }
 into empnum, empname;
onException
 myErrorHandler(6); // salir del programa
end

try
 get next from selectEmp;
onException
 if (sqlcode != 100)
 myErrorHandler(8); // salir del programa
 end
end

while (sqlcode != 100)
 empname = empname + " " + "III";

 try
 execute
 #sql{
 update employee
 set empname = :empname
 where current of selectEmp
 };
onException
 myErrorHandler(10); // salir del programa
end

 try
 get next from selectEmp;
onException
 if (sqlcode != 100)
 myErrorHandler(8); // salir del programa
 end
end
end // fin de while; el cursor se cierra automáticamente
// cuando se ha leído la última fila del conjunto de resultados

sysLib.commit();

```

En lugar de codificar las sentencias `get next` y `while`, puede utilizar la sentencia `forEach` que ejecuta un bloque de sentencias para cada fila de un conjunto de resultados:

```

VGVar.handleHardIOErrors = 1;

try
 open selectEmp
 with #sql{
 select empnum, empname
 from employee
 where empnum >= :empnum
 for update of empname
 }
 into empnum, empname;
onException
 myErrorHandler(6); // salir del programa
end

try
 forEach (from selectEmp)
 empname = empname + " " + "III";

 try
 execute
 #sql{
 update employee

```

```

 set empname = :empname
 where current of selectEmp
 };
 onException
 myErrorHandler(10); // salir del programa
 end
end // fin de forEach; el cursor se cierra automáticamente
// cuando se ha leído la última fila del conjunto de resultados

onException
// el bloque de excepción relacionado con forEach no se ejecuta si la condición
// es "sqlcode = 100", así que evite la prueba "if (sqlcode != 100)"
myErrorHandler(8); // salir del programa
end

sysLib.commit();

```

## Utilizar registros SQL con sentencias SQL implícitas

Para empezar a utilizar registros SQL EGL, declare un componente de registro SQL:

```

Record Employee type sqlRecord
{
 tableNames = [{"employee"}],
 keyItems = ["empnum"],
 defaultSelectCondition =
 #sqlCondition{
 aTableColumn = 4 -- iniciar cada componente SQL
 -- con un guión doble
 }
}

empnum decimal(6,0) {isReadOnly=yes};
empname char(40);
end

```

Declare un registro basado en el componente de registro:

```
emp Employee;
```

**Añadir una fila a una tabla SQL:** Para preparar la adición de una fila a una tabla SQL, coloque valores en el registro EGL:

```
emp.empnum = 1;
emp.empname = "John";
```

Añada un empleado a la tabla especificando la sentencia EGL add:

```

try
 add emp;
onException
 myErrorHandler(8);
end

```

**Leer filas de una tabla SQL:** Para preparar la lectura de filas de una tabla SQL, identifique una clave de registro:

```
emp.empnum = 1;
```

Obtenga una sola fila de cualquiera de estas formas:

- Especifique la sentencia EGL get de forma que genere una serie de sentencias (DECLARE cursor, OPEN cursor, FETCH row y, en ausencia de forUpdate, CLOSE cursor):

```

try
 get emp;
onException
 myErrorHandler(8);
end

```

- Especifique la sentencia EGL get de forma que genere una sola sentencia SELECT:

```

try
 get emp singleRow;
onException
 myErrorHandler(8);
end

```

Procese varias filas de cualquiera de estas formas:

- Utilice las sentencias EGL open, get next y while:

```

VGVar.handleHardIOErrors = 1;

```

```

try
 open selectEmp forUpdate for emp;
onException
 myErrorHandler(6); // salir del programa
end

```

```

try
 get next emp;
onException
 if (emp not noRecordFound)
 myErrorHandler(8); // salir del programa
 end
end

```

```

while (emp not noRecordFound)
 myRecord.empname = myRecord.empname + " " + "III";

```

```

try
 replace emp;
onException
 myErrorHandler(10); // salir del programa
end

```

```

try
 get next emp;
onException
 if (emp not noRecordFound)
 myErrorHandler(8); // salir del programa
 end
end

```

```

end // fin de while; el cursor se cierra automáticamente
// cuando se ha leído la última fila del conjunto de resultados

```

```

sysLib.commit();

```

- Utilice las sentencias EGL open y forEach:

```

VGVar.handleHardIOErrors = 1;

```

```

try
 open selectEmp forUpdate for emp;
onException
 myErrorHandler(6); // salir del programa
end

```

```

try
 forEach (from selectEmp)
 myRecord.empname = myRecord.empname + " " + "III";

```

```

 try
 replace emp;
 onException
 myErrorHandler(10); // salir del programa
 end
 end // fin de forEach; el cursor se cierra automáticamente
 // cuando se ha leído la última fila del conjunto de resultados

onException

 // el bloque de excepción relacionado con forEach no se ejecuta si la condición
 // es noRecordFound, así que evite la prueba "if (not noRecordFound)"
 myErrorHandler(8); // salir del programa
end

sysLib.commit();

```

## Utilizar registros SQL con sentencias SQL explícitas

Antes de utilizar registros SQL con sentencias SQL explícitas, debe declarar un componente de registro SQL. Este componente se diferencia del anterior en la sintaxis de las propiedades de elementos SQL y en la utilización de un valor calculado:

```

Record Employee type sqlRecord
{
 tableNameVariables = [{"empTable"}],
 // la utilización de una variable de nombre de tabla
 // significa que la tabla se especifica
 // durante la ejecución
 keyItems = ["empnum"]
}
empnum decimal(6,0) { isReadOnly = yes };
empname char(40);

// especificar propiedades de una columna calculada
aValue decimal(6,0)
{ isReadOnly = yes,
 column = "(empnum + 1) as NEWNUM" };
end

```

Declare variables:

```

emp Employee;
empTable char(40);

```

**Añadir una fila a una tabla SQL:** Para preparar la adición de una fila a una tabla SQL, coloque valores en el registro EGL y en una variable de nombre de tabla:

```

emp.empnum = 1;
emp.empname = "John";
empTable = "Employee";

```

Añada un empleado a la tabla especificando la sentencia EGL add y modificando la sentencia SQL:

```

// la variable de nombre de tabla no va precedida de un signo de dos puntos
try
 add emp
 with #sql{
 insert into empTable (empnum, empname)
 values (:empnum, :empname || ' ' || 'Smith')
 }

onException
 myErrorHandler(8);
end

```

**Leer filas de una tabla SQL:** Para preparar la lectura de filas de una tabla SQL, identifique una clave de registro:

```
emp.empnum = 1;
```

Obtenga una sola fila de cualquiera de estas formas:

- Especifique la sentencia EGL get de forma que genere una serie de sentencias (DECLARE cursor, OPEN cursor, FETCH row, CLOSE cursor):

```
try
 get emp into empname // La cláusula into es opcional. (No
 // no estar en la sentencia SELECT).
 with #sql{
 select empname
 from empTable
 where empnum = :empnum + 1
 }
onException
 myErrorHandler(8);
end
```

- Especifique la sentencia EGL get de forma que genere una sola sentencia SELECT:

```
try
 get emp singleRow // La cláusula into se deriva
 // del registro SQL y se basa en
 // las columnas de la cláusula select
 with #sql{
 select empname
 from empTable
 where empnum = :empnum + 1
 }
onException
 myErrorHandler(8);
end
```

Procese varias filas de cualquiera de estas formas:

- Utilice las sentencias EGL open, get next y while:

```
try
 // La cláusula into se deriva
 // del registro SQL y se basa en
 // las columnas de la cláusula select
 open selectEmp forUpdate
 with #sql{
 select empnum, empname
 from empTable
 where empnum >= :empnum
 order by NEWNUM -- utiliza el valor calculado
 for update of empname
 } for emp;
onException
 myErrorHandler(8); // salir del programa
end

try
 get next emp;
onException
 myErrorHandler(9); // salir del programa
end

while (emp not noRecordFound)
 try
 replace emp
 with #sql{
 update :empTable
```

```

 set empname = :empname || ' ' || 'III'
 } from selectEmp;

onException
 myErrorHandler(10); // salir del programa
end

try
 get next emp;
onException
 myErrorHandler(9); // salir del programa
end
end // fin de while

// no es necesario indicar "close emp;", ya que emp
// se cierra automáticamente cuando se ha leído
// el último registro del conjunto de resultados o
// (en caso de una excepción) cuando el programa finaliza

sysLib.commit();

```

- Utilice las sentencias EGL open y forEach:

```

try

 // La cláusula into se deriva
 // del registro SQL y se basa en
 // las columnas de la cláusula select
 open selectEmp forUpdate
 with #sql{
 select empnum, empname
 from empTable
 where empnum >= :empnum
 order by NEWNUM -- utiliza el valor calculado
 for update of empname
 } for emp;

onException
 myErrorHandler(8); // salir del programa
end

try
 forEach (from selectEmp)

 try
 replace emp
 with #sql{
 update :empTable
 set empname = :empname || ' ' || 'III'
 } from selectEmp;

onException
 myErrorHandler(9); // salir del programa
end

end // fin de sentencia forEach, y
// no es necesario indicar "close emp;", ya que emp
// se cierra automáticamente cuando se ha leído
// el último registro del conjunto de resultados o
// (en caso de una excepción) cuando el programa finaliza

onException
 // el bloque de excepción relacionado con forEach no se ejecuta si la condición
 // es noRecordFound, así que evite la prueba "if (not noRecordFound)"
 myErrorHandler(9); // salir del programa
end

sysLib.commit();

```

## Utilizar sentencias EGL prepare

Tiene la opción de utilizar un componente de registro SQL al codificar la sentencia EGL prepare. Declare el siguiente componente:

```
Record Employee type sqlRecord
{
 tableNames = [{"employee"}],
 keyItems = ["empnum"],
 defaultSelectCondition =
 #sqlCondition{
 aTableColumn = 4 -- iniciar cada componente SQL
 -- con un guión doble
 }
}

empnum decimal(6,0) {isReadOnly=yes};
empname char(40);
end
```

Declare variables:

```
emp Employee;
empnum02 decimal(6,0);
empname02 char(40);
myString char(120);
```

**Añadir una fila a una tabla SQL:** Antes de añadir una fila, asigne valores a las variables:

```
emp.empnum = 1;
emp.empname = "John";
empnum02 = 2;
empname02 = "Jane";
```

Desarrolle la sentencia SQL:

- Codifique la sentencia EGL prepare y haga referencia a un registro SQL, que proporciona una sentencia SQL que puede personalizarse:

```
prepare myPrep
 from "insert into employee (empnum, empname) " +
 "values (?, ?)" for emp;

// puede utilizar el registro SQL
// para comprobar el resultado de la operación
if (emp is error)
 myErrorHandler(8);
end
```

- Como alternativa, codifique la sentencia EGL prepare sin hacer referencia a un registro SQL:

```
myString = "insert into employee (empnum, empname) " +
 "values (?, ?)";

try
 prepare addEmployee from myString;
onException
 myErrorHandler(8);
end
```

En cada uno de los casos anteriores, la sentencia EGL prepare incluye espacios reservados para los datos que suministrará una sentencia EGL execute. A continuación se ofrecen dos ejemplos de la sentencia execute:

- Puede suministrar valores desde un registro (SQL u otro):  
execute addEmployee using emp.empnum, emp.empname;
- Puede suministrar valores desde elementos individuales:

```
execute addEmployee using empnum02, empname02;
```

**Leer filas de una tabla SQL:** Para preparar la lectura de filas de una tabla SQL, identifique una clave de registro:

```
empnum02 = 2;
```

Puede sustituir varias filas de cualquiera de estas formas:

- Utilice las sentencias EGL open, while y get next:

```
myString = "select empnum, empname from employee " +
 "where empnum >= ? for update of empname";
```

```
try
 prepare selectEmployee from myString for emp;
onException
 myErrorHandler(8); // salir del programa
end
```

```
try
 open selectEmp with selectEmployee
 using empnum02
 into emp.empnum, emp.empname;
onException
 myErrorHandler(9); // salir del programa
end
```

```
try
 get next from selectEmp;
onException
 myErrorHandler(10); // salir del programa
end
```

```
while (emp not noRecordFound)
```

```
 emp.empname = emp.empname + " " + "III";
```

```
 try
 replace emp
 with #sql{
 update employee
 set empname = :empname
 }
 from selectEmp;
 onException
 myErrorHandler(11); // salir del programa
end
```

```
 try
 get next from selectEmp;
 onException
 myErrorHandler(12); // salir del programa
end
```

```
end // fin de while; el cierre es automático cuando se ha leído la última fila
```

```
sysLib.commit();
```

- Utilice las sentencias EGL open y forEach:

```
myString = "select empnum, empname from employee " +
 "where empnum >= ? for update of empname";
```

```
try
 prepare selectEmployee from myString for emp;
onException
 myErrorHandler(8); // salir del programa
end
```

```

try
 open selectEmp with selectEmployee
 using empnum02
 into emp.empnum, emp.empname;
onException
 myErrorHandler(9); // salir del programa
end

try
 forEach (from selectEmp)
 emp.empname = emp.empname + " " + "III";

 try
 replace emp
 with #sql{
 update employee
 set empname = :empname
 }
 from selectEmp;
onException
 myErrorHandler(11); // salir del programa
end
end // end // fin de forEach; el cierre es automático cuando se ha leído la última fila
onException

// el bloque de excepción relacionado con forEach no se ejecuta si la condición
// es noRecordFound, así que evite la prueba "if (not noRecordFound)"
myErrorHandler(12); // salir del programa
end

sysLib.commit();

```

## Base de datos por omisión

La base de datos por omisión es una base de datos relacional a la que se accede cuando una sentencia de E/S relacionada con SQL se ejecuta en código generado por EGL y cuando no hay ninguna otra conexión de base de datos actual. La base de datos por omisión está disponible desde el principio de una unidad de ejecución; sin embargo, puede conectarse dinámicamente a una base de datos diferente para el acceso subsiguiente en la misma unidad de ejecución, como se describe en *VGLib.connectionService*.

La base de datos por omisión se especifica en la propiedad opcional de entorno de ejecución **vgj.jdbc.default.database**, que recibe un valor generado de la opción **sqlDB** del descriptor de construcción si la opción **genProperties** está establecida en GLOBAL o PROGRAM durante la generación.

### Conceptos relacionados

“Propiedades de tiempo de ejecución Java” en la página 347

“Unidad de ejecución” en la página 742

“Soporte de SQL” en la página 229

### Tareas relacionadas

“Establecer una conexión JDBC J2EE” en la página 362

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

“Cómo se realiza una conexión JDBC estándar” en la página 263

### Consulta relacionada

“Propiedades de ejecución de Java (detalles)” en la página 540

“sqlDB” en la página 400

“connectionService()” en la página 916

## Informix y EGL

Las normas siguientes son específicas de las bases de datos Informix y EGL:

- Una base de datos Informix a la que accede EGL o un programa generado por EGL debe tener las transiciones habilitadas.
- Si codifica una sentencia SQL y utiliza un signo de puntos (:) para identificar una tabla de Informix, utilice comillas para separar el identificador de Informix del resto de la sentencia, como en los ejemplos siguientes:

```
INSERT INTO "myDB:myTable"
(myColumn) values (:myField)
```

```
INSERT INTO "myDB@myServer:myTable"
(myColumn) values (:myField)
```

- Si utiliza la función de recuperación de SQL de EGL para acceder a los datos desde una base de datos Informix no ANSI, asegúrese de que las columnas de base de datos de tipo DECIMAL incluyan un valor de escala. En lugar de definir una columna como DECIMAL (4), por ejemplo, defínala como DECIMAL (4,0).
- Si tiene previsto utilizar la función de recuperación de SQL para recuperar datos de una tabla que forma parte de un esquema de sistemas Informix, debe establecer una preferencia especial, como se describe en el apartado *Establecer las preferencias de la recuperación de SQL*.

### Conceptos relacionados

"Soporte de SQL" en la página 229

### Tareas relacionadas

"Recuperar datos de tabla SQL"

"Establecer preferencias para la recuperación de SQL" en la página 121

---

## Tareas específicas de SQL

### Recuperar datos de tabla SQL

EGL proporciona un método para crear elementos de registro SQL a partir de la definición de una tabla SQL, vista o unión; para obtener una visión general, consulte *Soporte de SQL*.

Haga lo siguiente:

1. Asegúrese de que ha establecido las preferencias de SQL correctamente. Para conocer más detalles, consulte el apartado *Establecer preferencias para la recuperación de SQL*.
2. Decida dónde realizar la tarea:
  - En un archivo fuente de EGL, a medida que desarrolla cada registro de SQL; o bien
  - En la vista Esquema, que resultará más fácil cuando ya tenga registros de SQL.
3. Si está trabajando en el archivo fuente de EGL, haga lo siguiente:
  - a. Si no tiene el registro de SQL, créelo:
    - 1) Teclee **R**, pulse Control-espacio y en la lista de ayuda de contenido, seleccione una de las entradas de tabla SQL (normalmente **Registro SQL con nombres de tabla**).
    - 2) Teclee el nombre del registro SQL; pulse el tabulador y teclee un nombre de tabla, o una lista de tablas delimitada por comas, o el alias de una vista.

También puede crear un registro de SQL tecleando el contenido mínimo, apropiado si el nombre del registro es el mismo que el nombre de la tabla, como en este ejemplo:

```
Record myTable type sqlRecord
end
```

- b. Pulse con el botón derecho del ratón en cualquier parte del registro.
  - c. En el menú de contexto, pulse en **Registro SQL > Recuperar SQL**.
4. Si está trabajando en la vista Esquema, pulse con el botón derecho del ratón en la entrada para el registro de SQL y, en el menú de contexto, pulse en **Recuperar SQL**.

**Nota:** No puede recuperar una vista SQL que esté definida con la condición de DB2 WITH CHECK OPTIONS.

Tras crear elementos de registro, puede interesarle obtener beneficios de productividad creando los componentes dataItem equivalentes; consulte *Visión general de la creación de componentes dataItem a partir de un componente de registro de SQL*.

#### **Conceptos relacionados**

“Crear componentes dataItem de componente de reg. SQL (visión general)”  
“Soporte de SQL” en la página 229

#### **Tareas relacionadas**

“Crear componentes dataItem a partir de componente registro SQL” en la página 254  
“Establecer preferencias para conexiones a bases de datos SQL” en la página 119  
“Establecer preferencias para la recuperación de SQL” en la página 121

#### **Consulta relacionada**

“Propiedades de elementos SQL” en la página 67

## **Crear componentes dataItem de componente de reg. SQL (visión general)**

Después de declarar elementos de estructura en un componente de registro SQL, puede utilizar un mecanismo especial del editor EGL para crear componentes de elemento de datos que sean equivalentes a los elementos de estructura. La ventaja es que podrá crear más fácilmente un registro no SQL (normalmente un registro básico) para transferir datos a y desde el registro SQL relacionado durante la ejecución.

Considere los siguientes elementos de estructura:

```
10 myHostVar01 CHAR(3);
10 myHostVar02 BIN(9,2);
```

Puede solicitar que se creen componentes dataItem:

```
DataItem myHostVar01 CHAR(3) end

DataItem myHostVar02 BIN(9,2) end
```

Otro efecto es que las declaraciones de elemento de estructura se vuelven a escribir:

```
10 myHostVar01 myHostVar01;
10 myHostVar02 myHostVar02;
```

Como se muestra en este ejemplo, cada componente `dataItem` recibe el mismo nombre que el elemento de estructura relacionado y actúa como una `typedef` para el elemento de estructura. Cada componente de elemento de datos también está disponible como una `typedef` para otros elementos de estructura.

Antes de poder utilizar un elemento de estructura como base de un componente `dataItem`, el elemento de estructura debe tener un nombre, debe tener características primitivas válidas y no debe señalar a una `typedef`.

#### **Conceptos relacionados**

“Soporte de SQL” en la página 229

#### **Tareas relacionadas**

“Crear componentes `dataItem` a partir de componente registro SQL”

#### **Consulta relacionada**

“Componente `DataItem` en formato fuente EGL” en la página 472

“Componente de registro SQL en formato fuente EGL” en la página 748

### **Crear componentes `dataItem` a partir de componente registro SQL**

Después de declarar elementos de estructura en un componente de registro SQL, puede utilizar un mecanismo especial del editor EGL para crear componentes `dataItem` que sean equivalentes a los elementos de estructura. Para obtener información general, consulte *Visión general de la creación de componentes `dataItem` a partir de un componente de registro SQL*.

Si la vista Esquema no se visualiza, ábrala seleccionando **Mostrar vista > Esquema** en el menú Ventana.

En la vista Esquema haga lo siguiente:

1. Para un componente de registro SQL dado, mantenga pulsada **Control** mientras pulsa con el ratón en cada uno de los elementos de estructura que le interesen. Para seleccionar todos los elementos de estructura de un registro dado, pulse con el ratón en el elemento de estructura situado más arriba y mantenga pulsada la tecla **Mayús** mientras pulsa con el ratón en el elemento de estructura situado más abajo.
2. Pulse con el botón derecho del ratón en los elementos de estructura seleccionados.
3. En el menú de contexto, pulse en **Crear componente `DataItem`**.

Los componentes de elementos de datos se graban al final del archivo fuente EGL y cada elemento de estructura se modifica para hacer referencia al componente equivalente.

#### **Conceptos relacionados**

“Crear componentes `dataItem` de componente de reg. SQL (visión general)” en la página 253

“Soporte de SQL” en la página 229

#### **Tareas relacionadas**

“Recuperar datos de tabla SQL” en la página 252

#### **Consulta relacionada**

“Componente de registro SQL en formato fuente EGL” en la página 748

## Crear componentes de datos de EGL a partir de tablas de bases de datos relacionales

### Asistente Componentes de datos de EGL

El asistente Componentes de datos de EGL permite crear componentes de registros SQL así como componentes de elementos de datos relacionados y componentes de funciones basadas en biblioteca de una o varias tablas de bases de datos relacionales o de vistas preexistentes.

Después de conectar con la base de datos, puede hacer lo siguiente:

- Especifique los campos de clave de registro SQL que se utilizan para crear, leer, actualizar o suprimir una fila de una tabla o vista de base de datos dada.
- Personalice las sentencias SQL explícitas para crear, leer o actualizar una fila. (La sentencia SQL para suprimir una fila no puede personalizarse.)
- Especifique los campos de clave de registro SQL que se utilizan para seleccionar un conjunto de filas de una base de datos o vista dada.
- Personalice una sentencia SQL explícita para seleccionar un conjunto de filas.
- Valide y ejecute cada sentencia SQL

La salida incluye estos archivos:

- Un archivo fuente EGL que define cada componente de registro
- Una biblioteca EGL para cada componente de registro
- Un archivo fuente EGL que contiene todos los componentes de elemento de datos a los que hacen referencia los elementos de estructura en los componentes de registro SQL

Puede reducir el número de archivos si marca el recuadro de selección **Registro y biblioteca en el mismo archivo**.

### Conceptos relacionados

“Soporte de SQL” en la página 229

### Tareas relacionadas

“Crear, editar o suprimir conexión base datos para asistentes EGL” en la página 257

“Crear componentes datos EGL de tablas de bases datos relacionales”

“Personalizar sentencias SQL en los asistentes de EGL” en la página 257

## Crear componentes datos EGL de tablas de bases datos relacionales

Para crear componentes de datos de EGL a partir de tablas de bases de datos relacionales sin crear una aplicación Web aparte, haga lo siguiente:

1. Seleccione **Archivo > Nuevo >Otros...** Se visualiza un diálogo para seleccionar un asistente.
2. Expanda **EGL** y efectúe una doble pulsación sobre **Componentes de datos EGL**. Se visualiza el diálogo Componentes de datos EGL.
3. Especifique un nombre de proyecto EGL o Web EGL o seleccione un proyecto existente de la lista desplegable. Los componentes se generarán en este proyecto.
4. Seleccione una conexión de base de datos existente de la lista desplegable o establezca una nueva:
  - Para establecer una conexión de base de datos nueva, pulse **Añadir** e interactúe con el asistente *Conexión de base de datos nueva*. Para conocer más

detalles acerca de la clase de entrada necesaria para un campo determinado, pulse el botón derecho del ratón en el campo y pulse F1.

- Para obtener detalles sobre la edición o supresión de una conexión de base de datos, consulte la sección *Crear, editar o suprimir una conexión de base de datos para los asistentes de EGL*.

Cuando se establece una conexión con la base de datos, se visualiza una lista de tablas de base de datos.

5. Si desea aceptar el nombre del archivo EGL por omisión para elementos de datos, teclee un nombre de archivo nuevo.
6. En el campo **Seleccione los datos**, pulse el nombre de la tabla cuyas columnas le ayudarán a declarar componentes de datos. Para seleccionar varias tablas, mantenga pulsada la tecla **Ctrl** mientras pulsa distintos nombres de tabla. Para transferir el nombre o los nombres resaltados a la lista de tablas seleccionadas, pulse la flecha derecha.
7. Para cada una de las tablas seleccionadas (a la derecha), especifique el nombre del registro EGL a crear o acepte el nombre por omisión. Para eliminar una o varias tablas de esta lista, resalte las entradas de interés y pulse la flecha izquierda.
8. Si desea incluir el componente de biblioteca y los componentes de registro de SQL en el mismo archivo, marque el recuadro de selección.
9. Pulse en **Siguiente**.
10. Hay una pestaña disponible para cada tabla. En cada pestaña, seleccione el campo de clave que se utilizará al leer, actualizar y suprimir filas individuales y pulse la flecha derecha. Para seleccionar varios campos de clave, mantenga pulsada la tecla **Ctrl** mientras pulsa sobre distintos nombres de campo. Para eliminar un campo de clave de la lista de la derecha, resalte el nombre del campo y pulse la flecha izquierda.
11. Elija el campo de condición de selección que se utilizará al seleccionar un conjunto de filas y pulse la flecha derecha. Para seleccionar varios campos, mantenga pulsada la tecla **Ctrl** mientras pulsa sobre distintos nombres de campo. Para eliminar un campo de la lista de la derecha, resalte el nombre del campo y pulse la flecha izquierda.
12. Para personalizar una sentencia SQL implícita, consulte la sección *Personalizar sentencias SQL en los asistentes EGL*. Esta opción no está disponible para la sentencia delete de EGL.
13. Pulse en **Siguiente**.
14. Se visualiza la pantalla Generar componentes de datos EGL, incluyendo (en la parte inferior) una lista de los archivos que se generarán:
  - a. Para cambiar el nombre del proyecto EGL que recibirá los componentes EGL, teclee un nombre de proyecto en el campo Proyecto destino o seleccione un proyecto de la lista desplegable relacionada.
  - b. Para especificar los paquetes EGL para un tipo de componente específico (datos o biblioteca), teclee un nombre de paquete en el campo relacionado o seleccione un nombre de la lista desplegable relacionada.
15. Pulse en **Finalizar**.

### Conceptos relacionados

“Asistente Componentes de datos de EGL” en la página 255

“Asistente Páginas y componentes de datos de EGL” en la página 187

“Soporte de SQL” en la página 229

### Tareas relacionadas

“Crear una aplicación Web EGL de tabla única” en la página 188

“Crear, editar o suprimir conexión base datos para asistentes EGL”

“Personalizar sentencias SQL en los asistentes de EGL”

**Crear, editar o suprimir conexión base datos para asistentes EGL:** Cuando está en la primera pantalla de un asistente EGL para crear componentes de datos a partir de una base de datos relacional o para crear una aplicación Web a partir de una tabla de base de datos relacional, hay dos maneras de especificar una conexión de base de datos:

- Seleccionar una conexión existente de una lista desplegable
- Interactuar con el *asistente Conexión de base de datos nueva*.

Para utilizar ese asistente para crear una conexión, pulse **Añadir** y añada información según sea necesario. Para conocer más detalles acerca de la clase de entrada necesaria para un campo determinado, pulse el botón derecho del ratón en el campo y pulse F1.

Para editar una conexión de base de datos existente, haga lo siguiente:

1. Seleccione **Ventana > Abrir perspectiva > Otras**. En el diálogo Seleccionar perspectiva, marque el recuadro de selección **Mostrar todo** y efectúe una doble pulsación sobre **Datos**.
2. En la vista Explorador de base de datos, pulse con el botón derecho del ratón sobre la conexión de base de datos y seleccione **Editar conexión**. Revise las páginas del asistente de conexión de base de datos y cambie la información según convenga. Por ejemplo, pulse F1.
3. Para completar la edición, pulse **Finalizar**.

Para suprimir una conexión de base de datos existente, haga lo siguiente:

1. Seleccione **Ventana > Abrir perspectiva > Otras**. En el diálogo Seleccionar perspectiva, marque el recuadro de selección **Mostrar todo** y efectúe una doble pulsación sobre **Datos**.
2. En la vista Explorador de base de datos, pulse con el botón derecho del ratón sobre la conexión de base de datos y seleccione **Suprimir**.

### Conceptos relacionados

“Asistente Componentes de datos de EGL” en la página 255

“Asistente Páginas y componentes de datos de EGL” en la página 187

“Soporte de SQL” en la página 229

### Tareas relacionadas

“Crear una aplicación Web EGL de tabla única” en la página 188

“Crear componentes datos EGL de tablas de bases datos relacionales” en la página 255

“Establecer preferencias de EGL” en la página 113

**Personalizar sentencias SQL en los asistentes de EGL:** Cuando esté utilizando un asistente EGL para crear componentes de datos a partir de una tabla relacional o para crear una aplicación Web a partir de una tabla relacional, puede cambiar la sentencia SQL asociada a una acción como leer o actualizar:

1. Seleccione una acción de la lista Acciones de edición y pulse **Editar SQL**.
2. Edite la sentencia SQL (posible para todas las acciones excepto para delete) y pulse **Validar**. La validación asegura que la sentencia tenga la sintaxis correcta

y se ajuste a las reglas de los nombres de variable de lenguaje principal. Si la sentencia contiene errores, se visualiza un mensaje. Corrija los errores y vuelva a validar.

**Revertir a último** devuelve la sentencia a la última versión modificada válida. Las versiones anteriores no estarán disponibles después de cerrar el diálogo.

3. Pulse **Ejecutar** y vuelva a pulsar **Ejecutar**.
4. Si la sentencia SQL necesita valores para variables de lenguaje principal, se visualiza el diálogo Especificar valores de variable. Efectúe una doble pulsación sobre el campo **Valor** para especificar el valor de una variable de lenguaje principal y pulse la tecla **Intro**. Cuando haya entrado valores para todas las variables de lenguaje principal, pulse **Finalizar**.

**Nota:** Para las variables de lenguaje principal definidas como tipo *character*, debe encerrar el valor entre comillas.

5. Cuando haya terminado de ejecutar la sentencia SQL, pulse **Cerrar**.
6. Cuando haya terminado de editar las sentencias SQL, pulse **Aceptar**.

#### Conceptos relacionados

- “Asistente Componentes de datos de EGL” en la página 255
- “Asistente Páginas y componentes de datos de EGL” en la página 187
- “Soporte de SQL” en la página 229

#### Tareas relacionadas

- “Crear una aplicación Web EGL de tabla única” en la página 188
- “Crear componentes datos EGL de tablas de bases datos relacionales” en la página 255
- “Crear, editar o suprimir conexión base datos para asistentes EGL” en la página 257
- “Establecer preferencias de EGL” en la página 113

## Ver la sentencia SQL SELECT para un registro SQL

EGL proporciona una sentencia SQL SELECT implícita para un componente de registro SQL dado. Para ver la sentencia SQL SELECT implícita, haga lo siguiente:

1. Abra el archivo EGL que contiene el componente de registro SQL. Si no tiene el archivo abierto, en el Explorador de proyectos pulse con el botón derecho del ratón en el archivo de EGL y seleccione **Abrir con > Editor EGL**.
2. Pulse dentro del componente de registro SQL y, a continuación, pulse con el botón derecho del ratón. Aparecerá un menú de contexto.
3. Seleccione **Registro SQL > Ver selección por omisión**.
4. Para validar la sentencia SQL SELECT ante una base de datos, pulse en **Validar**.

**Nota:** Antes de utilizar la función de validar, los usuarios de DB2 UDB deben establecer la opción DEFERREDPREPARE. Puede establecer esta opción interactivamente en el CLP (procesador de línea de mandatos DB2) utilizando el mandato **db2 update cli cfg for section COMMON using DEFERREDPREPARE 0**. Este mandato colocará la palabra clave bajo la sección COMMON. Ejecute el mandato **db2 get cli cfg for section common** para verificar que se recoge la palabra clave.

#### Conceptos relacionados

- “Soporte de SQL” en la página 229

#### Tareas relacionadas

- “Validar la sentencia SQL SELECT para un registro SQL” en la página 259

#### Consulta relacionada

“Componente de registro SQL en formato fuente EGL” en la página 748

## Validar la sentencia SQL SELECT para un registro SQL

EGL proporciona una sentencia SQL SELECT implícita para un componente de registro SQL dado. Para validar la sentencia SQL SELECT implícita ante una base de datos, haga lo siguiente:

1. Abra el archivo EGL que contiene el componente de registro SQL. Si no tiene el archivo abierto, en el Explorador de proyectos pulse con el botón derecho del ratón en el archivo de EGL y seleccione **Abrir con > Editor EGL**.
2. Pulse dentro del componente de registro SQL y, a continuación, pulse con el botón derecho del ratón. Aparecerá un menú de contexto.
3. Seleccione **Registro SQL > Validar selección por omisión**.

**Nota:** Antes de utilizar la función de validar, los usuarios de DB2 UDB deben establecer la opción DEFERREDPREPARE. Puede establecer esta opción interactivamente en el CLP (procesador de línea de mandatos DB2) utilizando el mandato **db2 update cli cfg for section COMMON using DEFERREDPREPARE 0**. Este mandato colocará la palabra clave bajo la sección COMMON. Ejecute el mandato **db2 get cli cfg for section common** para verificar que se recoge la palabra clave.

#### Conceptos relacionados

“Soporte de SQL” en la página 229

#### Tareas relacionadas

“Ver la sentencia SQL SELECT para un registro SQL” en la página 258

#### Consulta relacionada

“Componente de registro SQL en formato fuente EGL” en la página 748

## Construir una sentencia EGL prepare

Dentro de una función, puede construir las especies de sentencias EGL siguientes basadas en un componente de registro SQL:

- Una sentencia EGL **prepare**
- La sentencia EGL relacionada **execute**, **open**, o **get**.

Haga lo siguiente:

1. Abra un archivo de EGL con el editor de EGL. El archivo debe contener una función y una sentencia SQL codificada. Si no tiene un archivo abierto, en el Explorador de proyectos del entorno de trabajo pulse con el botón derecho del ratón en el archivo de EGL y seleccione **Abrir con > Editor EGL**.
2. Pulse dentro de la función en la ubicación en la que residirá la sentencia EGL **prepare** y, a continuación, pulse con el botón derecho del ratón. Aparecerá un menú de contexto.
3. Seleccione **Añadir sentencia SQL Prepare**.
4. Teclee un nombre para identificar la sentencia EGL **prepare**. Para conocer las reglas, consulte *Convenios de denominación*.
5. Si tiene definida una variable de registro SQL, selecciónela en la lista desplegable. Aparecerá el nombre de componente de registro SQL correspondiente. Si no tiene definida una variable de registro SQL, puede teclear un nombre en el campo de nombre de variable de registro SQL y, a

continuación, seleccione un nombre de componente de registro SQL utilizando el botón **Examinar**. Más adelante deberá definir una variable de registro SQL con ese nombre en el código fuente EGL.

6. Seleccione un tipo de sentencia de ejecución en la lista desplegable.
7. Si la sentencia de ejecución es de tipo abierto, entre un identificador del conjunto de resultados.
8. Pulse en **Aceptar**. Las sentencias EGL se construyen dentro de la función.

#### Conceptos relacionados

“Soporte de SQL” en la página 229

#### Tareas relacionadas

“Validar la sentencia SQL SELECT para un registro SQL” en la página 259

“Ver la sentencia SQL SELECT para un registro SQL” en la página 258

#### Consulta relacionada

“Convenios de denominación” en la página 672

“Componente de registro SQL en formato fuente EGL” en la página 748

## Construir una sentencia SQL explícita a partir de una implícita

EGL proporciona una sentencia SQL implícita para cada sentencia de entrada/salida (E/S) EGL relacionada con SQL. Para construir una sentencia SQL explícita a partir de una implícita, haga lo siguiente:

1. Abra el archivo EGL que contiene la sentencia de E/S de EGL. Si no tiene el archivo abierto, en el Explorador de proyectos pulse con el botón derecho del ratón en el archivo de EGL y seleccione **Abrir con > Editor EGL**.
2. Pulse en la sentencia de E/S de EGL y, a continuación, pulse con el botón derecho del ratón. Aparecerá un menú de contexto.
3. Para construir una sentencia SQL explícita sin una cláusula INTO, seleccione **Sentencia SQL > Añadir**. Para construir una sentencia SQL explícita con una cláusula INTO, seleccione **Sentencia SQL > Añadir con Into**. La sentencia SQL implícita se añade a la sentencia de E/S de EGL convirtiéndola en una sentencia SQL explícita.

**Nota:** La cláusula INTO solamente es válida con sentencias **open**, **get** y **get next**.

#### Conceptos relacionados

“Soporte de SQL” en la página 229

#### Tareas relacionadas

“Eliminar una sentencia SQL de una sentencia EGL relacionada con SQL” en la página 262

“Restablecer una sentencia SQL explícita” en la página 261

“Validar una sentencia SQL implícita o explícita” en la página 261

“Ver la sentencia SQL para una sentencia EGL relacionada con SQL”

## Ver la sentencia SQL para una sentencia EGL relacionada con SQL

EGL proporciona una sentencia SQL implícita para cada sentencia de entrada/salida (E/S) EGL relacionada con SQL. Para ver la sentencia SQL implícita para una sentencia de E/S de EGL, haga lo siguiente:

1. Abra el archivo EGL que contiene la sentencia de E/S de EGL. Si no tiene el archivo abierto, en el Explorador de proyectos pulse con el botón derecho del ratón en el archivo de EGL y seleccione **Abrir con > Editor EGL**.

2. Pulse en la sentencia de E/S de EGL y, a continuación, pulse con el botón derecho del ratón. Aparecerá un menú de contexto.
3. Seleccione **Sentencia SQL > Ver**.

#### Conceptos relacionados

“Soporte de SQL” en la página 229

#### Tareas relacionadas

“Construir una sentencia SQL explícita a partir de una implícita” en la página 260  
 “Eliminar una sentencia SQL de una sentencia EGL relacionada con SQL” en la página 262  
 “Restablecer una sentencia SQL explícita”  
 “Validar una sentencia SQL implícita o explícita”

### Validar una sentencia SQL implícita o explícita

Para validar una sentencia SQL implícita o explícita ante una base de datos, haga lo siguiente:

1. Abra el archivo EGL que contiene la sentencia de EGL relacionada con SQL o la sentencia SQL explícita. Si no tiene el archivo abierto, en el Explorador de proyectos pulse con el botón derecho del ratón en el archivo de EGL y seleccione **Abrir con > Editor EGL**.
2. Pulse en la sentencia de EGL o la sentencia SQL y, a continuación, pulse con el botón derecho del ratón. Aparecerá un menú de contexto.
3. Seleccione **Sentencia SQL > Validar**.

**Nota:** Antes de utilizar la función de validar, los usuarios de DB2 UDB deben establecer la opción DEFERREDPREPARE. Puede establecer esta opción interactivamente en el CLP (procesador de línea de mandatos DB2) utilizando el mandato **db2 update cli cfg for section COMMON using DEFERREDPREPARE 0**. Este mandato colocará la palabra clave bajo la sección COMMON. Ejecute el mandato **db2 get cli cfg for section common** para verificar que se recoge la palabra clave.

#### Conceptos relacionados

“Soporte de SQL” en la página 229

#### Tareas relacionadas

“Construir una sentencia SQL explícita a partir de una implícita” en la página 260  
 “Eliminar una sentencia SQL de una sentencia EGL relacionada con SQL” en la página 262  
 “Restablecer una sentencia SQL explícita”  
 “Ver la sentencia SQL para una sentencia EGL relacionada con SQL” en la página 260

### Restablecer una sentencia SQL explícita

EGL proporciona una sentencia SQL implícita para cada sentencia de entrada/salida (E/S) EGL relacionada con SQL. Una sentencia SQL implícita puede añadirse a una sentencia de E/S de EGL convirtiéndola en una sentencia SQL explícita. Si cambia la sentencia SQL explícita, haga lo siguiente para volver a una sentencia SQL basada en la SQL implícita:

1. Abra el archivo EGL que contiene la sentencia SQL explícita. Si no tiene el archivo abierto, en el Explorador de proyectos pulse con el botón derecho del ratón en el archivo de EGL y seleccione **Abrir con > Editor EGL**.
2. Pulse en la sentencia SQL explícita y, a continuación, pulse con el botón derecho del ratón. Aparecerá un menú de contexto.
3. Seleccione **Sentencia SQL > Restablecer**.

### Conceptos relacionados

“Soporte de SQL” en la página 229

### Tareas relacionadas

“Construir una sentencia SQL explícita a partir de una implícita” en la página 260

“Eliminar una sentencia SQL de una sentencia EGL relacionada con SQL”

“Validar una sentencia SQL implícita o explícita” en la página 261

“Ver la sentencia SQL para una sentencia EGL relacionada con SQL” en la página 260

## Eliminar una sentencia SQL de una sentencia EGL relacionada con SQL

EGL proporciona una sentencia SQL implícita para cada sentencia de entrada/salida (E/S) EGL relacionada con SQL. Una sentencia de SQL implícita puede añadirse a una sentencia de E/S de EGL convirtiéndola en una sentencia SQL explícita (consulte *Construir una sentencia SQL explícita a partir de una implícita*). Para eliminar la sentencia SQL añadida, haga lo siguiente:

1. Abra el archivo EGL que contiene la sentencia SQL explícita. Si no tiene el archivo abierto, en el Explorador de proyectos pulse con el botón derecho del ratón en el archivo de EGL y seleccione **Abrir con > Editor EGL**.
2. Pulse en la sentencia SQL explícita y, a continuación, pulse con el botón derecho del ratón. Aparecerá un menú de contexto.
3. Seleccione **Sentencia SQL > Eliminar**. La sentencia de E/S de EGL no se elimina.

### Conceptos relacionados

“Soporte de SQL” en la página 229

### Tareas relacionadas

“Construir una sentencia SQL explícita a partir de una implícita” en la página 260

“Restablecer una sentencia SQL explícita” en la página 261

“Validar una sentencia SQL implícita o explícita” en la página 261

“Ver la sentencia SQL para una sentencia EGL relacionada con SQL” en la página 260

## Resolver una referencia para visualizar una sentencia SQL implícita

Considere lo que ocurre cuando se especifica la siguiente sentencia EGL:

```
open myRecord;
```

Cuando el editor EGL intenta crear una sentencia SQL por omisión, el editor intenta encontrar una variable llamada myRecord e identificar el componente de registro SQL en el que se basa dicha variable. Si la variable no está disponible durante el desarrollo o si la variable no está declarada, el editor intenta utilizar un componente de registro SQL llamado myRecord como base para la sentencia SQL por omisión. El editor supone que usted intenta crear una variable cuyo nombre es el nombre del componente de registro SQL.

Si desea almacenar una función relacionada con SQL en un archivo que no incluye la variable myRecord, puede realizar lo siguiente:

1. En el componente de programa, declare la variable global
2. Cree la función como una función anidada del componente de programa
3. Cree la sentencia SQL por omisión y modifíquela según convenga; a continuación, guarde el archivo

#### 4. Mueva la función al otro archivo

Una vez que la función se ha movido desde el componente de programa, el nombre de registro no puede resolverse durante el desarrollo y el editor no puede visualizar las sentencias SQL por omisión que se basan en dicho registro.

#### Conceptos relacionados

“Soporte de SQL” en la página 229

## Cómo se realiza una conexión JDBC estándar

Una conexión JDBC estándar se crea automáticamente durante la ejecución si está depurando un programa Java generado y si el archivo de propiedades del programa incluye los valores necesarios. Para conocer detalles sobre el significado de las propiedades del programa, incluidos detalles sobre cómo se derivan los valores, consulte *Propiedades de tiempo de ejecución de Java (detalles)*.

La conexión JDBC se basa en las siguientes clases de información:

#### URL de conexión

Si el código intenta acceder a una base de datos antes de invocar la función del sistema `sysLib.connect` o `VGLib.connectionService`, la URL de conexión es el valor de la propiedad `vgj.jdbc.default.database`.

Si el código intenta acceder a una base de datos como respuesta a una invocación de la función del sistema `sysLib.connect` o `VGLib.connectionService`, la URL de conexión es el valor de la propiedad `vgj.jdbc.databaseSN`.

Encontrará los detalles sobre el formato de una URL de conexión en *sqlValidationConnectionURL*.

#### ID de usuario

Si el código intenta acceder a una base de datos antes de invocar la función del sistema `sysLib.connect` o `VGLib.connectionService`, el ID de usuario es el valor de la propiedad `vgj.jdbc.default.userid`.

Si el código intenta acceder a una base de datos como respuesta a una invocación de una de esas funciones del sistema, el ID de usuario es un valor especificado en la invocación.

#### Contraseña

Si el código intenta acceder a una base de datos antes de invocar la función del sistema `sysLib.connect` o `VGLib.connectionService`, la contraseña es el valor de la propiedad `vgj.jdbc.default.password`.

Si el código intenta acceder a una base de datos como respuesta a una invocación de una de esas funciones del sistema, la contraseña es un valor especificado en la invocación. Puede utilizar una función del sistema para evitar exponer la contraseña en el archivo de propiedades del programa.

#### Clase de controlador JDBC

La clase de controlador JDBC es el valor de la propiedad `vgj.jdbc.drivers`.

#### Conceptos relacionados

“Archivo de propiedades del programa” en la página 350

#### Tareas relacionadas

“Establecer una conexión JDBC J2EE” en la página 362

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

### Consulta relacionada

"connect()" en la página 895  
"connectionService()" en la página 916  
"genProperties" en la página 394  
"Propiedades de ejecución de Java (detalles)" en la página 540  
"Requisitos de controlador JDBC en EGL" en la página 560  
"sqlDB" en la página 400  
"sqlID" en la página 401  
"sqlPassword" en la página 403  
"sqlValidationConnectionURL" en la página 403  
"sqlJDBCClass" en la página 401

---

## Soporte de VSAM

El soporte de VSAM para el código Java generado por EGL es el siguiente:

- El código basado en AIX puede acceder a los archivos VSAM locales
- El siguiente código puede acceder a los archivos VSAM remotos en z/OS:
  - El código Java generado por EGL que se ejecuta en Windows 2000/NT/XP
  - El depurador EGL, que se ejecuta en Windows 2000/NT/XP

## Requisitos previos de acceso

El acceso requiere que primero defina el archivo VSAM en el sistema donde desea que resida el archivo. El acceso remoto desde Windows 2000/NT/XP (tanto si es para el depurador EGL como durante la ejecución) también requiere que instale Distributed File Manager (DFM) en la estación de trabajo como se indica a continuación:

1. Localice el siguiente archivo en el directorio de instalación de EGL:  
`workbench\bin\VSAMWIN.zip`
2. Descomprima por zip el archivo en un nuevo directorio y siga las instrucciones del archivo INSTALL.README.

## Nombre de sistema

Para acceder a un archivo VSAM local, especifique el nombre de sistema en el componente de asociaciones de recursos y utilice el convenio de denominación adecuado para el sistema operativo. Para acceder a un archivo VSAM remoto desde el depurador EGL o desde código Java generado por EGL, especifique el nombre de sistema de la siguiente manera:

`\nombreMáquina\calificador.nombreArchivo`

*nombreMáquina*

El nombre de alias de LU SNA tal como se ha especificado en la configuración SNA

*calificador.nombreArchivo*

El nombre de conjunto de datos VSAM, incluido un calificador

El convenio de denominación es similar al formato del convenio de denominación universal (UNC). Para obtener detalles sobre el formato UNC, consulte la *Guía del usuario de Distributed FileManager*, que se encuentra en el siguiente archivo del directorio de instalación de EGL:

`workbench\bin\VSAMWIN.zip`

---

## Soporte de MQSeries

EGL da soporte al acceso de colas de mensajes MQSeries en cualquiera de las plataformas destino. Puede proporcionar este acceso de una de las siguientes maneras:

- Utilice palabras clave EGL relacionadas con MQSeries, como por ejemplo **add** y **get next**, en un registro MQ; en este caso, EGL oculta los detalles de MQSeries para que pueda centrarse en el problema empresarial que el código está tratando
- Invoque funciones EGL que llaman directamente a mandatos MQSeries, en cuyo caso están disponibles algunos mandatos que no están soportados por las palabras clave EGL

Puede mezclar ambos métodos en un determinado programa. Sin embargo, en la mayoría de los casos se utiliza exclusivamente uno de los dos métodos.

Independientemente del método que elija, puede controlar varias condiciones de ejecución personalizando los *registros de opciones*, que son registros básicos globales que los servicios de ejecución EGL pasan en las llamadas a MQSeries. Cuando declare un registro de opciones como variable de programa, puede utilizar un componente de registro de opciones instalado por EGL como una typedef; o bien puede copiar el componente instalado en el propio archivo EGL, personalizar el componente y utilizar el componente personalizado como una typedef.

El método elegido determina cómo los servicios de ejecución EGL hacen que los registros de opciones estén disponibles en MQSeries:

- Si trabaja con las sentencias EGL **add** y **get next**, identifique los registros de opciones cuando especifique las propiedades de un registro MQ. Si no identifica un determinado registro de opciones, EGL utiliza un valor por omisión.
- Si invoca las funciones EGL que llaman directamente a MQSeries, utilice los registros de opciones como argumentos al invocar las funciones. En este caso, no se dispone de valores por omisión.

Para obtener información detallada sobre los registros de opciones y sobre los valores que se pasan a MQSeries por omisión, consulte la sección *Registros de opciones para registros MQ*. Para obtener información detallada sobre el propio MQSeries, consulte los siguientes documentos:

- *An Introduction to Messaging and Queueing* (GC33-0805-01)
- *MQSeries MQI Technical Reference* (SC33-0850)
- *MQSeries Application Programming Guide* (SC33-0807-10)
- *MQSeries Application Programming Reference* (SC33-1673-06)

## Conexiones

Se conectará a un gestor de colas (llamado *gestor de colas de conexión*) la primera vez que invoque una sentencia de la lista siguiente:

- Una sentencia EGL **add** o **get next** que accede a una cola de mensajes
- Una invocación de la función EGL MQCONN o MQCONNX

Sólo puede acceder a un gestor de colas de conexión a la vez; sin embargo, puede acceder a varias colas que están bajo el control del gestor de colas de conexión. Si desea conectarse directamente a un gestor de colas que no sea el gestor de colas de conexión actual, debe desconectarse del primer gestor de colas invocando MQDISC y luego conectarse al segundo gestor de colas invocando **add**, **get next**, MQCONN o MQCONNX.

También puede acceder a colas que están bajo el control de un *gestor de colas remoto*, que es un gestor de colas con el que el gestor de colas de conexión puede interactuar. El acceso entre los dos gestores de colas sólo es posible si el propio MQSeries está configurado para permitir dicho acceso.

El acceso al gestor de colas de conexión se interrumpe cuando se invoca MQDISC o cuando finaliza el código.

## Incluir mensaje en transacción

Puede incorporar sentencias de acceso a colas en una unidad de trabajo de modo que todos los cambios realizados en las colas se comprometan o se retrotraigan en un único punto del proceso. Si una sentencia está una unidad de trabajo, se aplica lo siguiente:

- Una sentencia EGL **get next** (o una invocación EGL MQGET) sólo elimina un mensaje cuando se produce un compromiso
- El mensaje colocado en una cola utilizando una sentencia EGL **add** (o una invocación EGL MQPUT) sólo está visible fuera de la unidad de trabajo cuando se produce un compromiso

Cuando las sentencias de acceso a colas no están en una unidad de trabajo, cada cambio realizado en una cola de mensajes se compromete inmediatamente.

Una sentencia EGL **add** o **get next** relacionada con MQSeries se incorpora en una unidad de trabajo si la propiedad **includeMsgInTransaction** está en vigor para el registro MQ. El código generado incluye las siguientes opciones:

- Para MQGET, MQGMO\_SYNCPOINT
- Para MQPUT, MQPMO\_SYNCPOINT

Si no especifica la propiedad **includeMsgInTransaction** para un registro MQ, las sentencias de acceso a colas se ejecutan fuera de una unidad de trabajo. El código generado incluye las siguientes opciones:

- Para MQGET, MQGMO\_NO\_SYNCPOINT
- Para MQPUT, MQPMO\_NO\_SYNCPOINT

Cuando el código finaliza una unidad de trabajo, EGL compromete o retrotrae *todos* los recursos recuperables a los que accede el programa, que incluyen bases de datos, colas de mensajes y archivos recuperables. Este resultado se produce tanto si se utilizan las funciones de sistema (**sysLib.commit**, **sysLib.rollback**) como si se utilizan las llamadas EGL a MQSeries (MQCMIT, MQBACK); en cualquier caso, se invoca la función de sistema EGL apropiada.

Se produce una retrotracción si un programa EGL se termina prematuramente debido a un error detectado por los servicios de ejecución EGL.

## Personalización

Si desea personalizar la interacción con MQSeries en lugar de depender del proceso por omisión de las sentencias **add** and **get next**, lea la información siguiente.

### Componente EGL dataTable

Existe un conjunto componentes EGL dataTable que le ayudarán a interactuar con MQSeries. Cada componente permite a las funciones proporcionadas por EGL recuperar valores de listas basadas en memoria durante la ejecución. La sección siguiente incluye detalles sobre cómo se despliegan las tablas de datos.

## Cómo hacer posible la personalización

Para hacer posible la personalización, debe incorporar en el proyecto varios de los archivos EGL instalados *sin cambiarlos en modo alguno*. Los archivos son los siguientes:

### **records.egl**

Contiene componentes de registros básicos que pueden utilizarse como typedef para los registros de opciones que se utilizan en el programa; también incluye componentes de estructura que utilizan dichos registros y que proporcionan la flexibilidad de desarrollar componentes de registro propios

### **functions.egl**

Contiene dos conjuntos de funciones:

- Funciones de mandato MQSeries, que acceden directamente a MQSeries
- Funciones de inicialización, que permiten colocar valores iniciales en los registros de opciones que se utilizan en el programa

### **mqrcode.egl, mqrc.egl, mqvalue.egl**

Contiene un conjunto de componentes EGL dataTable que utilizan las funciones de mandato e inicialización

Las tareas son las siguientes:

1. Utilizando el proceso para importar archivos en el entorno de trabajo, incorpore dichos archivos en un proyecto EGL. Los archivos residen en el siguiente directorio:

```
dirInstalación\egl\eclipse\plugins\
com.ibm.etools.egl.generators_versión\MqReusableParts
```

#### *dirInstalación*

El directorio de instalación del producto, como por ejemplo C:\Program Files\IBM\RSPD\6.0. Si instaló y tuvo un producto de Rational Developer antes de instalar el producto que está utilizando ahora, deberá especificar el directorio utilizado en la instalación anterior.

#### *versión*

La última versión del conector; por ejemplo, 6.0.0

2. Para hacer que los componentes estén disponibles más fácilmente en el programa, escriba una o más sentencias *EGL import* en el archivo que contiene el programa. Si los archivos que deben importarse residen en un proyecto que no es el proyecto en el que está desarrollando el código, asegúrese de que su proyecto hace referencia al otro proyecto.  
Para obtener información detallada, consulte la sección *Import*.
3. En el programa, declare las siguientes variables globales:
  - Declare MQRC, MQRCODE y MQVALUE, cada una de las cuales debe utilizar como una typedef el componente dataTable que tiene el mismo nombre que la variable.
  - Para cada registro de opciones que desee pasar a MQSeries, declare un registro básico que utilice un componente de registro de opciones como una typedef. Para obtener información detallada sobre cada componente, consulte la sección *Registros de opciones para registros MQ*.
4. En la función, inicialice los registros de opciones que pretende pasar a MQSeries. Puede hacerlo fácilmente invocando la función de inicialización EGL importada para un determinado registro de opciones. El nombre de cada función es el nombre del componente que se utiliza como una typedef para el registro, seguido de *\_INIT*. Un ejemplo es MQGMO\_INIT.

5. Establezca valores en los registros de opciones. En muchos casos, se establece un valor asignando un símbolo EGL que representa una constante, cada una de las cuales se basa en un símbolo descrito en la documentación de MQSeries. Puede especificar varios símbolos EGL sumando los símbolos individuales, como en el siguiente ejemplo:

```
MQGMO.GETOPTIONS = MQGMO_LOCK
 + MQGMO_ACCEPT_TRUNCATED_MSG
 + MQGMO_BROWSE_FIRST
```

6. La primera vez que genere un determinado programa que utiliza las características de personalización del soporte de MQSeries, también generará las tablas de datos que utiliza dicho programa. Para generar todas las tablas de datos que se utilizan en el programa, deje que la opción del descriptor de construcción **genTables** tome por omisión el valor YES. Para obtener detalles adicionales, consulte la sección Componente DataTable.

## Palabras clave EGL relacionadas con MQSeries

Cuando se trabaja con las palabras clave EGL relacionadas con MQSeries, como por ejemplo *add* y *scan*, se define un registro MQ para cada cola de mensajes a la que se desea acceder. El diseño del registro es el formato del mensaje.

La tabla siguiente muestra las palabras clave.

| Palabra clave | Finalidad                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| add           | <p>Coloca el contenido de un registro MQ al final de la cola especificada.</p> <p>La sentencia add de EGL invoca hasta tres mandatos de MQSeries:</p> <ul style="list-style-type: none"> <li>• MQCONN conecta el código generado con un gestor de colas y se invoca cuando no hay ninguna conexión activa.</li> <li>• MQOPEN establece una conexión con una cola y se invoca cuando una conexión está activa pero la cola no está abierta.</li> <li>• MQPUT pone el registro en la cola y siempre se invoca a menos que se produzca un error en una llamada anterior de MQSeries.</li> </ul> <p>Después de añadir un registro MQ, debe cerrar una cola de mensajes antes de leer un registro MQ de la misma cola.</p> |
| close         | <p>Abandona el acceso a la cola de mensajes que está asociada a un registro MQ.</p> <p>La sentencia EGL close invoca el mandato MQCLOSE de MQSeries, que también se invoca automáticamente cuando finaliza el programa.</p> <p>Debe cerrar la cola de mensajes después de ejecutar una sentencia add o scan si otro programa necesita acceder a la cola. La sentencia close es especialmente apropiada si el programa se ejecuta durante mucho tiempo y el acceso ya no es necesario.</p>                                                                                                                                                                                                                             |

| Palabra clave | Finalidad                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| scan          | <p>Lee el primer mensaje de una cola en un registro de la cola de mensajes y (por omisión) elimina el mensaje de la cola.</p> <p>La sentencia scan de EGL invoca hasta tres mandatos de MQSeries:</p> <ul style="list-style-type: none"> <li>• MQCONN conecta el código generado con un gestor de colas y se invoca cuando no hay ninguna conexión activa.</li> <li>• MQOPEN establece una conexión con una cola y se invoca cuando una conexión está activa pero la cola no está abierta.</li> <li>• MQGET elimina el registro de la cola y se invoca siempre a menos que se haya producido un error en una llamada anterior a MQSeries.</li> </ul> <p>Después de leer un registro MQ, debe cerrar la cola antes de añadir un registro MQ a la misma cola.</p> |

## Especificación de cola y gestor

Cuando trabaje con palabras clave EGL relacionadas con MQSeries, debe identificar una cola en las siguientes situaciones:

- Durante la declaración, debe especificar un nombre de cola lógica estableciendo la propiedad **queueName** del componente de registro MQ. Este nombre de cola lógica actúa como valor por omisión para el nombre de cola al que se accede durante la ejecución; pero en la mayoría de casos, el nombre sólo es significativo como una forma de asociar el registro MQ con una cola física. El nombre de cola lógica no puede tener más de 8 caracteres.
- Durante la generación, debe controlar el proceso de generación con un componente buildDescriptor que, a su vez, puede hacer referencia a un componente de asociaciones de recursos. El componente de asociaciones de recursos asocia el nombre de cola con el nombre de una cola física.
- Durante la ejecución, el código puede cambiar el valor de la variable específica del registro **record.resourceAssociation** para alterar temporalmente cualquier nombre de cola que se haya especificado durante la declaración o la generación.

El nombre de la cola física tiene el siguiente formato:

*nombreGestorColas:nombreColaFísica*

*nombreGestorColas*

Nombre del gestor de colas; si se omite este nombre, también se omite el carácter de dos puntos

*nombreColaFísica*

Nombre de la cola física, como es conocido para el gestor de colas especificado

La primera vez que se emite una sentencia add o scan en un registro de cola de mensajes, se debe especificar un gestor de colas de conexión, ya sea por omisión o de otra forma. En el caso más simple, no se especifica ningún gestor de colas de conexión, pero se depende de un valor por omisión de la configuración de MQSeries.

La variable específica del registro **record.resourceAssociation** siempre contiene como mínimo el nombre de la cola de mensajes de un determinado registro MQ.

## Colas de mensajes remotas

Si desea acceder a una cola que está controlada por un gestor de colas remoto, debe realizar lo siguiente:

- Emita la sentencia EGL close para abandonar el acceso a la cola que se está utilizando ahora
- Establezca la variable específica del registro **record.resourceAssociation** para asegurar el acceso de la cola remota en el futuro

Establezca **record.resourceAssociation** de una de las dos maneras siguientes, en función de cómo estén establecidas las relaciones del gestor de colas en MQSeries:

- Si el gestor de colas de conexión tiene una definición local de la cola remota, establezca **record.resourceAssociation** de la manera siguiente:
  - Acepte el mismo valor para el gestor de colas de conexión (especificando el nombre del gestor de colas de conexión o no especificando ningún nombre; en este último caso, omita el carácter de dos puntos).
  - Especifique el nombre de la definición local de la cola remota.

El uso siguiente de la sentencia *add* o *scan* emite un mandato MQOPEN para establecer el acceso a la cola remota.

- Como alternativa, establezca **record.resourceAssociation** con el nombre del gestor de colas remoto, junto con el nombre de la cola remota. En este caso, el gestor de colas de conexión no cambia. El uso siguiente de la sentencia *add* o *scan* emite el mandato MQOPEN y utiliza la conexión ya establecida.

#### Conceptos relacionados

“Llamadas directas a MQSeries”

“Soporte de MQSeries” en la página 265

#### Consulta relacionada

“Propiedades de registros MQ” en la página 665

“Registros de opciones para registros MQ” en la página 665

## Llamadas directas a MQSeries

Puede utilizar un conjunto de funciones EGL instaladas que actúan como intermediario entre el código y MQSeries, como se describe en la sección *Soporte de MQSeries*.

La tabla siguiente muestra las funciones disponibles e identifica los argumentos necesarios. MQBACK (MQSTATE), por ejemplo, indica que cuando se invoca MQBACK se pasa un argumento basado en el componente de registro MQSTATE. Los componentes de registro se describen más adelante.

| Invocación de funciones EGL relacionadas con MQSeries | Efecto                                                                                                                                                                                                                                                            |
|-------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MQBACK (MQSTATE)                                      | Invoca la función de sistema sysLib.rollback para retrotraer una unidad lógica de trabajo. La retrotracción afecta a <i>todos</i> los recursos recuperables a los que accede el programa, que incluyen bases de datos, colas de mensajes y archivos recuperables. |
| MQBEGIN (MQSTATE, MQBO)                               | Inicia una unidad lógica de trabajo.                                                                                                                                                                                                                              |

| Invocación de funciones EGL relacionadas con MQSeries | Efecto                                                                                                                                                                                                                                                                                                                |
|-------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MQCHECK_COMPLETION (MQSTATE)                          | Establece el campo mqdescription del registro que se basa en MQSTATE. El valor se basa en el código de razón devuelto por última vez. La función MQCHECK_COMPLETION se llama automáticamente desde las funciones EGL MQBEGIN, MQCLOSE, MQCONN, MQCONNX, MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQPUT1 y MQSET.          |
| MQCLOSE (MQSTATE)                                     | Cierra la cola de mensajes a la que hace referencia MQSTATE.hobj.                                                                                                                                                                                                                                                     |
| MQCMIT (MQSTATE)                                      | Invoca la función de sistema sysLib.commit para comprometer una unidad lógica de trabajo. El compromiso afecta a <i>todos</i> los recursos recuperables a los que accede el programa, que incluyen bases de datos, colas de mensajes y archivos recuperables.                                                         |
| MQCONN (MQSTATE, nombreGestorc)                       | Conecta a un gestor de colas, que se identifica mediante <i>nombreGestorc</i> , una serie que puede tener hasta 48 caracteres. MQSeries establece el handle de conexión (MQSTATE.hconn) que se utilizará en las llamadas subsiguientes.<br><b>Nota:</b> El código puede conectarse a un gestor de colas a la vez.     |
| MQCONNX(MQSTATE, nombreGestorc, MQCNO)                | Conecta a un gestor de colas con opciones que controlan la forma en que funciona la llamada. El gestor de colas se identifica mediante <i>nombreGestorc</i> , una serie que puede tener hasta 48 caracteres. MQSeries establece el handle de conexión (MQSTATE.hconn) que se utilizará en las llamadas subsiguientes. |
| MQDISC (MQSTATE)                                      | Desconecta de un gestor de colas.                                                                                                                                                                                                                                                                                     |
| MQGET(MQSTATE, MQMD, MQGMO, BUFFER)                   | Lee y elimina un mensaje de la cola. El almacenamiento intermedio no puede tener más de 32767 bytes, pero esta restricción no se aplica si se utiliza la sentencia EGL get next.                                                                                                                                      |
| MQINQ(MQSTATE, MQATTRIBUTES)                          | Solicita atributos de una cola.                                                                                                                                                                                                                                                                                       |
| MQNOOP()                                              | Sólo lo utiliza EGL.                                                                                                                                                                                                                                                                                                  |
| MQOPEN(MQSTATE, MQOD)                                 | Abre una cola de mensajes. MQSeries establece el handle de cola (MQSTATE.hobj) que se utilizará en las llamadas subsiguientes.                                                                                                                                                                                        |
| MQPUT(MQSTATE, MQMD, MQPMO, BUFFER)                   | Añade un mensaje a la cola. El almacenamiento intermedio no puede tener más de 32767 bytes, pero esta restricción no se aplica si se utiliza la sentencia EGL <b>add</b> .                                                                                                                                            |
| MQPUT1(MQSTATE, MQOD, MQMD, MQPMO, BUFFER)            | Abre una cola, escribe un único mensaje y cierra la cola.                                                                                                                                                                                                                                                             |
| MQSET(MQSTATE, MQATTRIBUTES)                          | Establece atributos de una cola.                                                                                                                                                                                                                                                                                      |

La tabla siguiente muestra los registros de opciones que se utilizan como argumentos cuando se invocan las funciones EGL relacionadas con MQSeries. También se muestra la función de inicialización que debe invocarse para un determinado argumento.

El primer paso es inicializar el argumento que se basa en el componente de registro MQSTATE. En el ejemplo siguiente (como en la tabla que se muestra a continuación), se supone que el nombre del argumento es el mismo que el nombre del componente de registro:

```
MQSTATE_INIT(MQSTATE);
```

| Argumento (el nombre del componente de registro) | Función de inicialización   | Descripción                                                                                                                                                                                                                      |
|--------------------------------------------------|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MQATTRIBUTES                                     | ninguna                     | Matrices de atributos y selectores de atributos, además de otra información que se utiliza en el mandato MQINQ o MQSET                                                                                                           |
| MQBO                                             | MQBO_INIT (MQBO)            | Opciones de inicio                                                                                                                                                                                                               |
| MQCNO                                            | MQCNO_INIT (MQCNO)          | Opciones de conexión                                                                                                                                                                                                             |
| MQGMO                                            | MQGMO_INIT (MQGMO)          | Opciones de obtención de mensaje                                                                                                                                                                                                 |
| MQIIH                                            | MQIIH_INIT (MQIIH)          | Cabecera de información IMS; describe la información que se necesita al principio de un mensaje MQSeries enviado a IMS (la documentación de MQSeries indica que el uso de esta cabecera no está soportado en Windows 2000/NT/XP) |
| MQINTATTRS                                       | ninguna                     | Matrices de atributos de tipo entero que se utilizarán en el mandato MQINQ o MQSET                                                                                                                                               |
| MQMD                                             | MQMD_INIT (MQMD, MQSTATE)   | Descriptor de mensaje (MQSeries versión 2)                                                                                                                                                                                       |
| MQMDE                                            | MQMDE_INIT (MQMDE, MQSTATE) | Extensión de descriptor de mensaje (utilice solamente los campos que están en MQSeries versión 2)                                                                                                                                |
| MQOD                                             | MQOD_INIT (MQOD)            | Descriptor de objeto                                                                                                                                                                                                             |
| MQOO                                             | MQOO_INIT (MQOO)            | Opciones de apertura                                                                                                                                                                                                             |
| MQPMO                                            | MQPMO_INIT (MQPMO)          | Opciones de colocación de mensaje                                                                                                                                                                                                |
| MQSELECTORS                                      | ninguna                     | Una matriz de selectores de atributos, que sólo se utiliza si desea acceder a MQSeries sin utilizar las funciones EGL                                                                                                            |

| Argumento (el nombre del componente de registro) | Función de inicialización   | Descripción                                                                                                                                                                                                                                                                |
|--------------------------------------------------|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MQSTATE                                          | MQSTATE_INIT (MQSTATE)      | Una colección de argumentos cada uno de los cuales se utiliza en una o más llamadas a MQSeries; por ejemplo, cuando se conecta con la función EGL MQCONN o MQCONN, MQSeries establece el handle de conexión (MQSTATE.hconn) que se utilizará en las llamadas subsiguientes |
| MQXQH                                            | MQXQH_INIT (MQXQH, MQSTATE) | Cabecera de cola de transmisión                                                                                                                                                                                                                                            |

**Nota:** Cada componente de registro sólo contiene un elemento de estructura, y el elemento de estructura utiliza un componente de estructura como una typeDef. Esta configuración proporciona una gran flexibilidad. Puede crear sus propios componentes de registro cada uno de los cuales se compone de una serie de componentes de estructura.

El nombre de cada componente de estructura es el nombre del componente de registro seguido de \_S; el componente de registro MQGMO, por ejemplo, utiliza un componente de estructura llamado MQGMO\_S.

#### Conceptos relacionados

“Palabras clave EGL relacionadas con MQSeries” en la página 268

“Soporte de MQSeries” en la página 265

“Componentes de registro” en la página 132

“Typedef” en la página 27

#### Consulta relacionada

“get next” en la página 597

“commit()” en la página 893

“rollback()” en la página 905



---

## Mantener código EGL

---

### Línea de comentario de código fuente EGL

Para comentar una línea de código, haga lo siguiente:

1. Pulse en la línea y después pulse con el botón derecho del ratón. Se muestra un menú de contexto.
2. Seleccione **Comentar**. Se colocan indicadores de comentario (//) al principio de la línea.

Para comentar varias líneas consecutivas de código, haga lo siguiente:

1. Pulse la línea inicial. Mientras mantiene pulsado el botón izquierdo del ratón, arrastre el cursor hasta la línea final. Suelte el botón del ratón y el grupo de líneas quedará resaltado.
2. Pulse el botón derecho del ratón y seleccione **Comentar** en el menú de contexto. Se colocan indicadores de comentario (//) al principio de cada una de las líneas del grupo seleccionado.

Utilice los mismos procedimientos para descomentar líneas, pero seleccione **Descomentar** en el menú de contexto.

#### Tareas relacionadas

“Crear un archivo fuente EGL” en la página 128

“Abrir un componente en un archivo .egl” en la página 277

#### Consulta relacionada

“Editor EGL” en la página 483

---

## Buscar componentes

Si tiene un archivo abierto en el editor de EGL, puede buscar componentes después de establecer los criterios de búsqueda:

1. Abra un archivo EGL. No puede utilizar el recurso de búsqueda a menos que el editor de EGL esté activo; sin embargo, la búsqueda no se limita al archivo abierto en el editor.
2. En el menú Entorno de trabajo, pulse **Buscar > EGL**. Se visualiza el diálogo Buscar.
3. Si la pestaña Búsqueda EGL no está ya visualizada, pulse **Búsqueda EGL**. Tenga en cuenta que las condiciones especificadas en la pestaña Búsqueda pueden afectar al resultado.
4. Teclee el nombre de un componente que desee localizar; o para visualizar una lista de componentes con nombres que coincidan con un patrón de caracteres específico, incluya comodines dentro del nombre:
  - Un signo de interrogación (?) representa cualquier carácter
  - Un asterisco (\*) representa una serie de caracteres cualesquiera

Por ejemplo, teclee *myForm?Group* para localizar componentes denominados *myForm1Group* y *myForm2Group*, pero no *myForm10Group*. Teclee *myForm\*Group* para localizar componentes denominados *myForm1Group*, *myForm2Group* y *myForm10Group*.

5. Para que la búsqueda sea sensible a las mayúsculas/minúsculas (de forma que myFormGroup sea distinto de MYFORMGROUP), pulse el recuadro de selección.
6. En el recuadro Buscar, seleccione un tipo de componente o seleccione **Cualquier elemento** para ampliar la búsqueda a todos los tipos de componente.
7. En el recuadro Limitar a, seleccione la opción para limitar la búsqueda a declaraciones de componente, referencias de componente o a ambas.
8. En el recuadro Ámbito, seleccione **Área de trabajo** para buscar en el área de trabajo, **Proyectos delimitadores** para buscar en el proyecto resaltado actualmente en el Explorador de proyectos o **Conjunto de trabajo** para buscar en un conjunto de proyectos definido. Si elige el ámbito de Conjunto de trabajo, pulse el botón **Elegir** para seleccionar un conjunto de trabajo existente o para definir uno nuevo.
9. Pulse el botón **Buscar**. El resultado de la búsqueda se muestra en la vista Buscar.
10. Si efectúa una doble pulsación sobre un archivo en la vista Buscar, el archivo se abre en el editor de EGL y el componente coincidente aparece resaltado. Si hay más de una coincidencia en el archivo, aparecerá resaltada la primera coincidencia.  
Las flechas del margen izquierdo del editor indican las ubicaciones de cada componente coincidente.

#### Conceptos relacionados

“Componentes” en la página 17

#### Tareas relacionadas

“Abrir un componente en un archivo .egl” en la página 277

#### Consulta relacionada

“Editor EGL” en la página 483

---

## Ver referencias de componente

Puede visualizar una vista jerárquica de los componentes de EGL a los que se hace referencia en un componente de programa, biblioteca, PageHandler o manejador de informes y puede acceder a esos componentes:

1. Abra la vista Referencia de componentes de una de las dos formas siguientes:
  - En el Explorador de proyectos, pulse con el botón derecho del ratón sobre un archivo EGL que contenga un componente de programa, biblioteca, PageHandler o manejador de informes. Seleccione **Abrir en referencia de componentes**.
  - Además, abra un archivo de EGL en el editor de EGL:
    - a. Si la vista Esquema no se visualiza, ábrala seleccionando **Mostrar vista > Esquema** en el menú Ventana.
    - b. En la vista Esquema, pulse sobre un archivo con el botón derecho del ratón y seleccione **Abrir en referencia de componentes**.
2. El componente de programa, biblioteca, PageHandler o manejador de informes está en la parte superior de la jerarquía; cada componente con referencias es un subelemento de esa jerarquía y para cada componente la vista muestra parámetros, declaraciones de datos, declaraciones de uso y funciones, según convenga.

3. Efectúe una doble pulsación sobre un componente. El archivo fuente relacionado se abre en el editor EGL y el nombre de componente aparece resaltado.

**Conceptos relacionados**

“Proyectos, paquetes y archivos EGL” en la página 13  
“Componentes” en la página 17

**Tareas relacionadas**

“Localizar un archivo fuente EGL en el Explorador de proyectos” en la página 278  
“Abrir un componente en un archivo .egl”

**Consulta relacionada**

“Editor EGL” en la página 483

---

## Abrir un componente en un archivo .egl

Con unas pocas teclas puede acceder a un componente EGL que no sea un componente de construcción en cualquier lugar del área de trabajo:

1. En el área de trabajo, pulse **Navegar > Abrir componente** o pulse el botón **Abrir componente** de la barra de herramientas. Se visualizará el diálogo Abrir componente.
2. Teclee el nombre del componente que desee localizar; o para visualizar una lista de componentes con nombres que coincidan con un patrón de caracteres específico, incluya comodines dentro del nombre:
  - Un signo de interrogación (?) representa cualquier carácter
  - Un asterisco (\*) representa una serie de caracteres cualesquieraPor ejemplo, teclee *myForm?Group* para localizar componentes denominados *myForm1Group* y *myForm2Group*, pero no *myForm10Group*. Teclee *myForm\*Group* para localizar componentes denominados *myForm1Group*, *myForm2Group* y *myForm10Group*.  
Mientras teclea el nombre, los componentes que cumplen la condición se visualizan en el diálogo Abrir componente, en la sección Componentes coincidentes.
3. En la lista de componentes, seleccione el que desea abrir. La sección Calificador del diálogo visualiza la vía de acceso que contiene la carpeta, el proyecto, el paquete y el archivo fuente que mantiene el componente seleccionado. En caso de que múltiples componentes tengan el mismo nombre, seleccione un componente pulsando en la vía de acceso del archivo que desee abrir.
4. Pulse en **Aceptar**. El archivo fuente que contiene el componente que ha seleccionado se abrirá en el editor EGL con el nombre de componente resaltado.

**Conceptos relacionados**

“Proyectos, paquetes y archivos EGL” en la página 13  
“Componentes” en la página 17

**Tareas relacionadas**

“Crear un archivo fuente EGL” en la página 128  
“Localizar un archivo fuente EGL en el Explorador de proyectos” en la página 278

**Consulta relacionada**

“Editor EGL” en la página 483

---

## Localizar un archivo fuente EGL en el Explorador de proyectos

Si está editando un archivo fuente EGL, puede localizar el archivo rápidamente en la vista Explorador de proyectos. La opción del menú de contexto **Mostrar** en Explorador de proyectos hace lo siguiente:

- Abre la vista Explorador de proyectos, si no está abierta todavía
- Expande los nodos del árbol de Explorador de proyectos necesarios para localizar el archivo fuente
- Resalta el archivo fuente

Para localizar un archivo fuente EGL en el Explorador de proyectos, haga lo siguiente:

1. Pulse con el botón derecho del ratón dentro del área del editor de un archivo fuente EGL abierto. Aparecerá un menú de contexto.
2. Seleccione **Mostrar en Explorador de proyectos** en el menú de contexto.

### Tareas relacionadas

“Crear un archivo fuente EGL” en la página 128

“Abrir un componente en un archivo .egl” en la página 277

### Consulta relacionada

“Editor EGL” en la página 483

---

## Suprimir un archivo de EGL en el Explorador de proyectos

Para suprimir un archivo EGL en el Explorador de proyectos, haga lo siguiente:

1. Pulse el archivo EGL y pulse la tecla Suprimir. También puede pulsar el archivo EGL con el botón derecho del ratón y cuando se visualiza el menú de contexto, seleccione **Suprimir**.
2. Se le solicitará confirmación para suprimir el archivo. Pulse **Sí** para suprimir el archivo o **No** para cancelar la supresión.

### Tareas relacionadas

“Crear un archivo fuente EGL” en la página 128

“Localizar un archivo fuente EGL en el Explorador de proyectos”

---

## Depurar código EGL

---

### Depurador EGL

Cuando esté en el entorno de trabajo, el depurador EGL le permite depurar código EGL sin que primero tenga que generar la salida. Se aplican las siguientes categorías:

- Para depurar PageHandlers, así como programas que se utilizan en un contexto J2EE, puede utilizar el entorno de prueba local de WebSphere Application Server en modalidad de depuración:
  - Debe utilizar este entorno para todo el código que se ejecuta bajo J2EE en una aplicación Web.
  - Puede utilizar este entorno para programas que se ejecutan en una aplicación por lotes bajo J2EE.
- Para depurar otro código (aplicaciones por lotes que no se ejecutan bajo J2EE; o aplicaciones de texto), utilice una configuración de lanzamiento que esté fuera del entorno de prueba de WebSphere. En este caso, puede iniciar la sesión de depuración pulsando algunas teclas.

Si trabaja en un programa por lotes que intenta desplegar en un contexto J2EE, puede utilizar la configuración de lanzamiento para depurar el programa en un contexto no J2EE. Aunque la configuración es más simple, debe ajustar algunos valores:

- Debe establecer el valor la opción del descriptor de construcción J2EE en NO cuando utilice la configuración de lanzamiento.
- Además, debe ajustar los valores de propiedad Java para salvar las diferencias que existen al acceder a una base de datos relacional:
  - Para J2EE, especifique una serie como *jdbc/MyDB*, que es el nombre al que está enlazado un origen de datos en el registro JNDI. Especifique esta serie de las siguientes maneras:
    - Estableciendo la opción del descriptor de construcción *sqlJNDIName*; o bien
    - Especificando un valor en la página de preferencias Conexiones de base de datos SQL EGL en el campo Nombre JNDI de conexión; para obtener información detallada, consulte la sección *Establecer las preferencias de las conexiones de base de datos SQL*.
  - Para no J2EE, especifique un URL de conexión, como por ejemplo *jdbc:db2:MyDB*. Especifique esta serie de las siguientes maneras:
    - Estableciendo la opción del descriptor de construcción *sqlDB*; o bien
    - Especificando un valor en la página de preferencias Conexiones de base de datos SQL EGL, en el campo URL de conexión; para obtener información detallada, consulte la sección *Establecer las preferencias de las conexiones de base de datos SQL*.

Más adelante, una sección describe la interacción de los descriptors de construcción y las preferencias de EGL.

### Mandatos del depurador

Utilice los siguientes mandatos para interactuar con el depurador EGL:

### **Añadir punto de interrupción**

Identifica una línea en la que se interrumpe el proceso. Cuando se interrumpe la ejecución del código, puede examinar los valores de variable así como el estado de los archivos y pantallas.

Los puntos de interrupción se recuerdan de una sesión de depuración a la siguiente, a menos que elimine el punto de interrupción.

No se puede establecer un punto de interrupción en una línea en blanco ni en una línea de comentario.

### **Inhabilitar punto de interrupción**

Desactiva un punto de interrupción pero no lo elimina.

### **Habilitar punto de interrupción**

Activa un punto de interrupción que previamente se ha inhabilitado.

### **Eliminar punto de interrupción**

Borra el punto de interrupción de modo que el proceso ya no se interrumpe automáticamente en la línea.

### **Eliminar todos los puntos de interrupción**

Borra todos los puntos de interrupción.

### **Ejecutar**

Ejecuta el código hasta el siguiente punto de interrupción o hasta que finaliza la unidad de ejecución. (En cualquier caso, el depurador se detiene en la primera sentencia de la función principal.)

### **Ejecutar hasta línea**

Ejecuta todas las sentencias hasta (pero sin incluirla) la sentencia de una línea especificada.

### **Recorrer todo**

Ejecuta la siguiente sentencia EGL y se interrumpe.

La lista siguiente indica lo que ocurre si se emite el mandato **step into** para un determinado tipo de sentencia:

#### **call**

Se detiene en la primera sentencia de un programa llamado si éste se ejecuta en el depurador EGL. Se detiene en la siguiente sentencia del programa actual si el programa llamado se ejecuta fuera del depurador EGL.

El depurador EGL busca el programa receptor en cada uno de los proyectos del entorno de trabajo.

#### **converse**

Espera la entrada de usuario. Esta entrada hace que el proceso se detenga en la siguiente sentencia en ejecución, que puede estar en una función de validador.

#### **forward**

Si el código se reenvía a un PageHandler, el depurador espera la entrada de usuario y se detiene en la siguiente sentencia en ejecución, que puede estar en una función de validador.

Si el código se reenvía a un programa, el depurador se detiene en la primera sentencia de dicho programa.

#### **function invocation**

Se detiene en la primera sentencia de la función.

### **JavaLib.invoke y funciones relacionadas**

Se detiene en la siguiente sentencia Java, de modo que se puede depurar el código Java que está disponible mediante las funciones de acceso Java.

#### **show, transfer**

Se detiene en la primera sentencia del programa que recibe el control. El programa destino es el fuente EGL que se ejecuta en el depurador EGL y no es el código generado por EGL.

Después de una sentencia **show** o de una sentencia **transfer** del tipo *transferir a transacción*, el depurador EGL cambia al descriptor de construcción del nuevo programa, o bien (si este descriptor de construcción está siendo utilizado) solicita al usuario un nuevo descriptor de construcción. El nuevo programa puede tener un conjunto distinto de propiedades respecto del programa que se ha ejecutado anteriormente.

El depurador EGL busca el programa receptor en cada uno de los proyectos del entorno de trabajo.

#### **Recorrer principal**

Ejecuta la siguiente sentencia EGL y se interrumpe, pero no se detiene en las funciones que se invocan desde la función actual.

La lista siguiente indica lo que ocurre si se emite el mandato **step over** para un determinado tipo de sentencia:

#### **converse**

Espera la entrada de usuario y luego omite cualquier función de validación (a menos que esté en vigor un punto de interrupción). Se detiene en la sentencia que sigue a la sentencia **converse**.

#### **forward**

Si el código se reenvía a un PageHandler, el depurador espera la entrada de usuario y se detiene en la siguiente sentencia en ejecución, pero no en una función de validador, a menos que esté en vigor un punto de interrupción.

Si el código se reenvía a un programa, el depurador se detiene en la primera sentencia de dicho programa.

#### **show, transfer**

Se detiene en la primera sentencia del programa que recibe el control. El programa destino es el fuente EGL que se ejecuta en el depurador EGL y no es el código generado por EGL.

Después de una sentencia **show** o de una sentencia **transfer** del tipo *transferir a transacción*, el depurador EGL cambia al descriptor de construcción del nuevo programa, o bien (si este descriptor de construcción está siendo utilizado) solicita al usuario un nuevo descriptor de construcción. El nuevo programa puede tener un conjunto distinto de propiedades respecto del programa que se ha ejecutado anteriormente.

El depurador EGL busca el programa receptor en cada uno de los proyectos del entorno de trabajo.

#### **Recorrer hasta retorno**

Ejecuta las sentencias necesarias para regresar a una función o programa invocador; a continuación, se interrumpe en la sentencia que recibe el control en dicha función o programa.

Se aplica una excepción si se emite el mandato **step return** en una función de validador. En este caso, el comportamiento es idéntico al de un mandato **step into**, que básicamente significa que el depurador EGL ejecuta la siguiente sentencia y se interrumpe.

El depurador EGL trata las siguientes sentencias EGL como si fueran operadores nulos:

- **sysLib.audit**
- **sysLib.purge**
- **sysLib.startTransaction**

Por ejemplo, puede añadir un punto de interrupción en estas sentencias, pero un mandato **step into** simplemente continúa hasta la sentencia siguiente, que no tiene ningún otro efecto.

Finalmente, si emite el mandato **step into** o **step over** para una sentencia que es la última que se ejecuta en la función (y si dicha sentencia no es **return**, **exit program** o **exit stack**), el proceso se interrumpe en la propia función para que pueda revisar las variables que son locales a la función. Para continuar la sesión de depuración en este caso, emita otro mandato.

## Utilización de descriptores de construcción

Un descriptor de construcción ayuda a determinar aspectos del entorno de depuración. El depurador EGL selecciona el descriptor de construcción de acuerdo con las siguientes normas:

- Si ha especificado un descriptor de construcción de depuración para el programa o PageHandler, el depurador EGL utiliza dicho descriptor de construcción. Para obtener información detallada sobre cómo establecer el descriptor de construcción de depuración, consulte la sección *Establecer los descriptores de construcción por omisión*.
- Si no ha especificado un descriptor de construcción de depuración, el depurador EGL le solicita que seleccione uno en la lista de descriptores de construcción o que acepte el valor **None**. Si acepta el valor **None**, el depurador EGL construye un descriptor de construcción que se utilizará durante la sesión de depuración; y una preferencia determina si se aplica la compatibilidad con VisualAge Generator.
- Si ha especificado **None** o bien un descriptor de construcción que no tiene una parte de la información de conexión de base de datos necesaria, el depurador EGL obtiene la información de conexión revisando las preferencias. Para obtener los detalles sobre cómo establecer estas preferencias, consulte la sección *Establecer las preferencias de las conexiones de base de datos SQL*.

Si depura un programa que debe utilizarse en una aplicación de texto o por lotes de un entorno Java y si dicho programa emite una sentencia **transfer** que pasa el control a un programa que también debe utilizarse en una unidad de ejecución distinta de un entorno Java, el depurador EGL utiliza un descriptor de construcción que está asignado al programa receptor. La elección del descriptor de construcción se basa en las normas descritas anteriormente.

Si depura un programa que está llamado por otro programa, el depurador EGL utiliza el descriptor de construcción que está asignado al programa llamado. La elección del descriptor de construcción se basa en las normas descritas anteriormente, excepto que si no especifica un descriptor de construcción, el

depurador no le solicita uno cuando se invoca el programa llamado; en su lugar, se sigue utilizando el descriptor de construcción del programa llamante.

**Nota:** Debe utilizar un descriptor de construcción distinto para el llamador y el programa llamado si uno de estos programas (pero no ambos) utiliza la compatibilidad con VisualAge Generator. El estado en tiempo de generación de la compatibilidad con VisualAge está determinado por el valor de la opción del descriptor de construcción **VAGCompatibility**.

Un componente descriptor de construcción o de asociaciones de recursos que se utiliza para depurar código puede ser distinto del que se utiliza para generar código.

## Acceso a base de datos SQL

Para determinar el ID de usuario y la contraseña que deben utilizarse para acceder a una base de datos SQL, el depurador EGL considera las siguientes fuentes en orden hasta que se encuentra la información o bien se consideran todas las fuentes:

1. El descriptor de construcción utilizado durante la depuración; concretamente, las opciones del descriptor de construcción `sqlID` y `sqlPassword`.
2. La página de preferencias de SQL, como se describe en la sección *Establecer las preferencias de las conexiones de base de datos SQL*; en esta página, también se especifica otras informaciones sobre conexión.
3. Un diálogo interactivo que se visualiza durante la conexión. Este diálogo sólo se visualiza si marca el recuadro de selección **Solicitar ID de usuario y contraseña SQL cuando sea necesario**.

## Sentencia call

Como se ha indicado antes, el depurador EGL responde a una sentencia **transfer** o **show** interpretando el código fuente EGL. Sin embargo, el depurador EGL responde a una sentencia **call**, revisando el componente de opciones de enlace especificado en el descriptor de construcción, si existe. Si el componente de opciones de enlace al que se hace referencia incluye un elemento **callLink** para la llamada, el resultado es el siguiente:

- Si la propiedad **callLink remoteComType** está establecida en **DEBUG**, el depurador EGL interpreta el código fuente EGL. El depurador busca el fuente haciendo referencia a las propiedades **callLink package** y **location**.
- Si la propiedad **callLink remoteComType** no está establecida en **DEBUG**, el depurador invoca el código generado por EGL y utiliza la información del componente de opciones de enlace como si el depurador estuviera ejecutando un programa Java generado por EGL.

En ausencia de información de enlace, el depurador EGL responde a una sentencia **call** interpretando el código fuente EGL. La información de enlace no está disponible en los siguientes casos:

- No se utiliza ningún descriptor de construcción; o bien
- Se utiliza un descriptor de construcción, pero no se ha especificado ningún componente de opciones de enlace en dicho descriptor de construcción; o bien
- Se ha especificado un componente de opciones de enlace en el descriptor de construcción, pero el componente al que se hace referencia no tiene un elemento **callLink** que hace referencia al programa llamado.

Si el depurador ejecuta el código fuente EGL, puede ejecutar sentencias en dicho programa emitiendo el mandato **step into** desde el llamador. Sin embargo, si el

depurador llama al código generado, el depurador ejecuta todo el programa; el mandato **step into** funciona igual que el mandato **step over**.

## Tipo de sistema utilizado durante la depuración

Un valor para el tipo de sistema está disponible en `sysVar.systemType`. Además, un segundo valor está disponible en `VGLib.getVAGSysType` si ha solicitado la compatibilidad con VisualAge Generator durante el desarrollo.

El valor en `sysLib.systemType` es el mismo que el valor de la opción del descriptor de construcción `system`, excepto que el valor es `DEBUG` en uno de estos dos casos:

- Ha seleccionado la preferencia **Establecer systemType en DEBUG**, como se ha mencionado en la sección *Establecer las preferencias del depurador EGL*; o bien
- Ha especificado `NONE` como descriptor de construcción que debe utilizarse durante la sesión de depuración, sea cual sea el valor de dicha preferencia.

La función de sistema `VGLib.getVAGSysType` devuelve el equivalente de VisualAge Generator del valor de `sysLib.systemType`; para obtener información detallada, consulte la tabla de `VGLib.getVAGSysType`.

## Puerto del depurador EGL

El depurador EGL utiliza un puerto para establecer la comunicación con el entorno de trabajo Eclipse. El número de puerto por omisión es 8345. Si otra aplicación está utilizando ese puerto o si ese puerto está bloqueado por un cortafuegos, establezca un valor distinto tal como se describe en *Establecer preferencias para el depurador EGL*.

Si se especifica un valor que no sea 8345 como puerto de depurador EGL y si se va a depurar un programa EGL en el servidor J2EE, debe editar la configuración de servidor:

1. Vaya a la pestaña Entorno, sección Propiedades del sistema
2. Pulse Añadir
3. Para Nombre, escriba `com.ibm.debug.egl.port`
4. Para Valor, escriba el número de puerto

## Recomendaciones

Mientras prepara el trabajo con el depurador EGL, tenga en cuenta las siguientes recomendaciones (la mayoría de las cuales presuponen que la variable de sistema `sysVar.systemType` está establecida en `DEBUG` cuando se depura el código):

- Si está recuperando una fecha de una base de datos pero espera que el código de ejecución recupere dicha fecha en un formato distinto de ISO, escriba una función para convertir la fecha, pero invoque la función sólo cuando el tipo de sistema sea `DEBUG`. El formato ISO es `aaaa-mm-dd`, que es el único que está disponible en el depurador.
- Para especificar clases Java externas que se utilizarán cuando se ejecute el depurador, modifique la vía de acceso de clases, como se describe en la sección *Establecer las preferencias del depurador EGL*. Es posible que necesite clases adicionales para dar soporte, por ejemplo, a `MQSeries`, controladores `JDBC` o funciones de acceso Java.
- Cuando depure una función de `PageHandler` que ha sido invocada por JSF (y no por otra función EGL), utilice `Run` para salir de la función en lugar de `Step Over`, `Step Into` o `Step Return`. La utilización de cualquiera de los tres mandatos `Step` le lleva al código Java generado del `PageHandler`, que no es útil cuando se

depura EGL. Si utiliza uno de los mandatos Step, utilice Run para salir del código Java generado y visualizar la página Web en un navegador.

- Si utiliza la opción SQL WITH HOLD (o la opción equivalente de EGL), debe saber que la opción WITH HOLD no está disponible para Java generado por EGL ni en el depurador EGL. Puede evitar esta limitación, en parte colocando sentencias commit dentro de una sentencia condicional que sólo se invoca durante la ejecución, como en el siguiente ejemplo:

```
if (systemType not debug)
 sysLib.commit();
end
```

Si los programas EGL se depuran en el servidor J2EE o mediante un escucha EGL, el servidor o el escucha EGL deben configurarse para indicar el número correspondiente al puerto del depurador EGL:

- Para configurar un servidor J2EE, edite la configuración del servidor:
  1. Vaya a la pestaña Entorno, sección Propiedades del sistema
  2. Pulse **Añadir**
  3. En el campo **Nombre**, escriba *com.ibm.debug.egl.port*
  4. En el campo **Valor**, escriba el nuevo número de puerto
- Para configurar un escucha EGL, edite la configuración de lanzamiento del escucha EGL:
  1. Vaya a la pestaña Argumentos
  2. En el campo **Argumentos de VM**, escriba lo siguiente:  
*-Dcom.ibm.debug.egl.port=númeroPuerto*

*númeroPuerto*

El nuevo número de puerto

#### Conceptos relacionados

“Compatibilidad con VisualAge Generator” en la página 439

“Opciones de codificación de caracteres para el depurador EGL” en la página 115

“Soporte de VSAM” en la página 264

#### Tareas relacionadas

“Establecer preferencias para conexiones a bases de datos SQL” en la página 119

“Establecer preferencias para el depurador de EGL” en la página 114

“Establecer los descriptores de construcción por omisión” en la página 116

#### Consulta relacionada

“Propiedad remoteComType del elemento callLink” en la página 419

“sqlDB” en la página 400

“sqlID” en la página 401

“sqlJNDIName” en la página 402

“sqlPassword” en la página 403

“getVAGSysType()” en la página 919

“systemType” en la página 937

---

## Depurar aplicaciones que no sean J2EE

### Iniciar una aplicación no de J2EE en el depurador de EGL

Para iniciar la depuración de un programa de texto EGL o un programa básico no de J2EE en una sesión de depuración de EGL, es necesaria una configuración de lanzamiento. Una configuración de lanzamiento define la ubicación del archivo de

un programa y especifica cómo deberá lanzarse el programa. Puede dejar que la aplicación de EGL cree la configuración de lanzamiento (creación implícita), o bien puede crear una personalmente (consulte *Crear una configuración de lanzamiento en el depurador de EGL*).

Para lanzar un programa utilizando una configuración de lanzamiento creada implícitamente, haga lo siguiente:

1. En la vista Explorador de proyectos, pulse con el botón derecho del ratón en el archivo fuente de EGL que desee lanzar. Otra posibilidad es que si el archivo fuente EGL está abierto en el editor de EGL, puede pulsar con el botón derecho del ratón en el programa en la vista Esquema.
2. Aparecerá un menú de contexto.
3. Pulse en **Depurar programa EGL**. Se creará una configuración de lanzamiento y el programa se lanzará en el depurador de EGL.

Para ver la configuración de lanzamiento creada implícitamente, haga lo siguiente:

1. Pulse en la flecha junto al botón Depurar en la barra de herramientas. Aparecerá un menú de contexto.
2. Pulse en **Depurar**. Aparecerá el diálogo Depurar. El nombre de la configuración de lanzamiento se visualiza en el campo Nombre. Las configuraciones de lanzamiento creadas implícitamente se denominan de acuerdo con los nombres de proyecto y archivo fuente.

**Nota:** También puede visualizar el diálogo Depurar pulsando en **Depurar** en el menú Ejecutar.

#### Conceptos relacionados

“Depurador EGL” en la página 279

#### Tareas relacionadas

“Crear una configuración de lanzamiento en el depurador de EGL”

“Recorrer una aplicación en el depurador de EGL” en la página 291

“Utilizar puntos de interrupción en el depurador de EGL” en la página 290

“Ver variables en el depurador de EGL” en la página 292

## Crear una configuración de lanzamiento en el depurador de EGL

Para iniciar la depuración de un programa de texto EGL o un programa básico no de J2EE en una sesión de depuración de EGL, es necesaria una configuración de lanzamiento. Una configuración de lanzamiento define cómo deberá lanzarse el programa. Puede crear una configuración de lanzamiento (creación explícita), o bien puede dejar que la aplicación EGL cree una automáticamente (consulte *Iniciar un programa no de J2EE en el depurador de EGL*).

Para iniciar un programa utilizando una configuración de lanzamiento creada explícitamente, haga lo siguiente:

1. Pulse en la flecha junto al botón Depurar en la barra de herramientas y, a continuación, pulse en **Depurar**, o seleccione **Depurar** en el menú Ejecutar.
2. Se visualiza el diálogo Depurar.
3. Pulse en **Programa EGL** en la lista Configuraciones y, a continuación, pulse en **Nuevo**.
4. Si no tenía un archivo fuente EGL resaltado en la vista Explorador de proyectos, la configuración de lanzamiento se denomina *Configuración\_nueva*. Si

tenía un archivo fuente EGL resaltado en la vista Explorador de proyectos, la configuración de lanzamiento tiene el mismo nombre que el archivo fuente de EGL. Si desea cambiar el nombre de la configuración de lanzamiento, teclee el nuevo nombre en el campo Nombre.

5. Si el nombre en el campo Proyecto de la pestaña Cargar no es correcto, pulse en **Examinar**. Se visualiza una lista de proyectos. Pulse en un proyecto y, a continuación, pulse en **Aceptar**.
6. Si el nombre en el campo de archivo fuente de programa EGL no es correcto o el campo está vacío, pulse en **Buscar**. Se visualiza una lista de archivos fuente de EGL. Pulse en un archivo fuente y, a continuación, pulse en **Aceptar**.
7. Si ha realizado cambios en alguno de los campos del diálogo Depurar, pulse en **Aplicar** para guardar los valores de la configuración de lanzamiento.
8. Pulse en **Depurar** para lanzar el programa en el depurador de EGL.

**Nota:** Si todavía no ha utilizado **Aplicar** para guardar los valores de la configuración de lanzamiento, pulsando en **Revertir** se eliminarán todos los cambios que haya realizado.

#### Conceptos relacionados

“Depurador EGL” en la página 279

#### Tareas relacionadas

“Iniciar una aplicación no de J2EE en el depurador de EGL” en la página 285

“Recorrer una aplicación en el depurador de EGL” en la página 291

“Utilizar puntos de interrupción en el depurador de EGL” en la página 290

“Ver variables en el depurador de EGL” en la página 292

## Crear una configuración de lanzamiento de escucha de EGL

Para depurar una aplicación EGL no J2EE llamada desde una envoltura o una aplicación Java generada por EGL, se necesita una configuración de lanzamiento de escucha de EGL. Para crear una configuración de lanzamiento de escucha de EGL, haga lo siguiente:

1. Pulse en la flecha junto al botón Depurar en la barra de herramientas y, a continuación, pulse en **Depurar**, o seleccione **Depurar** en el menú Ejecutar.
2. Se visualiza el diálogo Depurar.
3. Pulse en **Escucha EGL** en la lista Configuraciones y, a continuación, pulse en **Nuevo**.
4. La configuración de lanzamiento de escucha se llama *New\_configuration*. Si desea cambiar el nombre de la configuración de lanzamiento, teclee el nuevo nombre en el campo Nombre.
5. Si no especifica un número de puerto, el valor por omisión del puerto es 8346; de lo contrario, especifique un número de puerto. Cada escucha de EGL necesita su propio puerto.
6. Pulse **Aplicar** para guardar la configuración de lanzamiento de escucha.
7. Pulse **Depurar** para lanzar la escucha de EGL.

#### Conceptos relacionados

“Depurador EGL” en la página 279

#### Tareas relacionadas

“Crear una configuración de lanzamiento en el depurador de EGL” en la página 286

“Iniciar una aplicación no de J2EE en el depurador de EGL” en la página 285

“Recorrer una aplicación en el depurador de EGL” en la página 291

“Utilizar puntos de interrupción en el depurador de EGL” en la página 290

“Ver variables en el depurador de EGL” en la página 292

---

## Depurar aplicaciones J2EE

### Preparar un servidor para la depuración Web EGL

Para depurar programas Web de EGL que se ejecutan en WebSphere Application Server, debe preparar el servidor para la depuración. El paso de preparación debe realizarse una vez por servidor y no es necesario volver a realizarlo, incluso si se concluye el entorno de trabajo.

Para preparar un servidor para depurar, haga lo siguiente:

1. Si está trabajando con el Entorno de prueba de WebSphere v5.1, asegúrese de que el servidor esté detenido. Si está trabajando con WebSphere Application Server v6.0, asegúrese de que el servidor esté ejecutándose. La explicación de esta diferencia es que el código v6.0 es un servidor en funcionamiento.
2. En la vista Servidor, pulse con el botón derecho del ratón en el servidor. Aparecerá un menú de contexto.
3. Seleccione **Habilitar/inhabilitar depuración de EGL**. Un mensaje indica que ha habilitado la depuración de EGL.
4. Si desea depurar el código Java generado en lugar de EGL, vuelva a pulsar con el botón derecho del ratón sobre el servidor y seleccione **Habilitar/inhabilitar la depuración de EGL**. Un mensaje indica que ha inhabilitado la depuración de EGL.

#### Conceptos relacionados

“Depurador EGL” en la página 279

“WebSphere Application Server y EGL” en la página 341 “Soporte Web” en la página 187

#### Tareas relacionadas

“Iniciar una sesión de depuración Web EGL” en la página 289

“Iniciar un servidor para la depuración Web EGL”

“Recorrer una aplicación en el depurador de EGL” en la página 291

“Utilizar puntos de interrupción en el depurador de EGL” en la página 290

“Ver variables en el depurador de EGL” en la página 292

### Iniciar un servidor para la depuración Web EGL

Si está trabajando con una aplicación Web basada en EGL que accede a un origen de datos JNDI, no puede seguir las instrucciones del tema actual a menos que anteriormente haya configurado un servidor de aplicaciones Web. Para obtener información específica de WebSphere, consulte *WebSphere Application Server y EGL*.

Además, si desea depurar un programa Web de EGL, debe prepara el servidor para el objetivo tal como se describe en *Preparar un servidor para la depuración Web de EGL*.

Para iniciar el servidor para la depuración, haga lo siguiente:

1. En la vista Servidor, pulse con el botón derecho del ratón sobre el servidor
2. Seleccione **Depurar > Depurar en servidor**

#### Conceptos relacionados

“Depurador EGL” en la página 279

“WebSphere Application Server y EGL” en la página 341 “Soporte Web” en la página 187

### Tareas relacionadas

- “Preparar un servidor para la depuración Web EGL” en la página 288
- “Iniciar una sesión de depuración Web EGL”
- “Recorrer una aplicación en el depurador de EGL” en la página 291
- “Utilizar puntos de interrupción en el depurador de EGL” en la página 290
- “Ver variables en el depurador de EGL” en la página 292

## Iniciar una sesión de depuración Web EGL

Si está trabajando con una aplicación Web basada en EGL que accede a un origen de datos JNDI, no puede seguir las instrucciones del tema actual a menos que anteriormente haya configurado un servidor de aplicaciones Web. Para obtener información específica de WebSphere, consulte *WebSphere Application Server y EGL*.

Además, si desea depurar un programa Web de EGL, debe preparar el servidor para el objetivo tal como se describe en *Preparar un servidor para la depuración Web de EGL*. Ahorrará tiempo en el procedimiento actual si ya ha iniciado el servidor para depurar, tal como se describe en la sección *Iniciar un servidor para la depuración Web de EGL*.

Para iniciar una sesión de depuración Web EGL, haga lo siguiente:

1. En el Explorador de proyectos, expanda las carpetas **WebContent** y **WEB-INF**. Pulse con el botón derecho del ratón sobre el archivo JSP que desea ejecutar y seleccione **Depurar > Depurar en servidor**. Se visualiza el diálogo Selección de servidor.
2. Si ya ha configurado un servidor para este proyecto Web, seleccione **Elegir un servidor existente** y seleccione un servidor de la lista. Pulse **Finalizar** para iniciar el servidor (si es necesario), para desplegar la aplicación en el servidor y para iniciar la aplicación.
3. Si no ha configurado un servidor para este proyecto Web, puede continuar de la manera siguiente, pero solo si la aplicación no accede a un origen de datos JNDI:
  - a. Seleccione **Definir manualmente un servidor**.
  - b. Especifique el nombre de sistema principal que (para el sistema local) es **localhost**.
  - c. Seleccione un tipo de servidor que sea parecido al servidor de aplicaciones Web en el que pretende desplegar la aplicación en tiempo de ejecución. Las opciones incluyen **Entorno de prueba de WebSphere v5.1** y **WebSphere v6.0 Server**.
  - d. Si no pretende cambiar las opciones al trabajar en el proyecto actual, marque el recuadro de selección para **Establecer servidor como proyecto predeterminado**.
  - e. En la mayoría de los casos puede ahorrarse este paso, pero si desea especificar valores distintos de los predeterminados, pulse **Siguiente** y haga sus selecciones.
  - f. Pulse **Finalizar** para iniciar el servidor, para desplegar la aplicación en el servidor y para iniciar la aplicación.

### Conceptos relacionados

- “Depurador EGL” en la página 279
- “WebSphere Application Server y EGL” en la página 341
- “Soporte Web” en la página 187

### Tareas relacionadas

- “Preparar un servidor para la depuración Web EGL” en la página 288

- “Iniciar un servidor para la depuración Web EGL” en la página 288
- “Recorrer una aplicación en el depurador de EGL” en la página 291
- “Utilizar puntos de interrupción en el depurador de EGL”
- “Ver variables en el depurador de EGL” en la página 292

---

## Utilizar puntos de interrupción en el depurador de EGL

Los puntos de interrupción se utilizan para la pausa en la ejecución de un programa. Puede gestionar puntos de interrupción dentro o fuera de una sesión de depuración de EGL. Tenga en cuenta lo siguiente al trabajar con puntos de interrupción:

- Una marca de color azul en el margen izquierdo de la vista Fuente indica que se ha establecido y habilitado un punto de interrupción.
- Una marca de color blanco en el margen izquierdo de la vista Fuente indica que se ha establecido un punto de interrupción pero está inhabilitado.
- La ausencia de una marca en el margen izquierdo indica que no se ha establecido un punto de interrupción.

### Añadir o eliminar un punto de interrupción

Añada o elimine un único punto de interrupción en un archivo fuente EGL realizando una de las siguientes acciones:

- Sitúe el cursor en la línea de punto de interrupción en el margen izquierdo de la vista Fuente y realice una doble pulsación.
- Sitúe el cursor en la línea de punto de interrupción en el margen izquierdo de la vista Fuente y pulse con el botón derecho del ratón. Aparecerá un menú de contexto. Pulse en el elemento de menú adecuado.

### Habilitar o inhabilitar un punto de interrupción

Habilite o inhabilite un único punto de interrupción en un archivo fuente EGL realizando las siguientes acciones:

1. En la vista Punto de interrupción, pulse con el botón derecho del ratón en el punto de interrupción. Aparecerá un menú de contexto.
2. Pulse en el elemento de menú adecuado.

### Eliminar todos los puntos de interrupción

Elimine todos los puntos de interrupción de un archivo fuente EGL realizando las siguientes acciones:

1. Pulse con el botón derecho del ratón en cualquiera de los puntos de interrupción visualizados en la vista Puntos de interrupción. Aparecerá un menú de contexto.
2. Pulse en **Eliminar todo**.

### Conceptos relacionados

“Depurador EGL” en la página 279

### Tareas relacionadas

- “Crear una configuración de lanzamiento en el depurador de EGL” en la página 286
- “Iniciar una aplicación no de J2EE en el depurador de EGL” en la página 285
- “Recorrer una aplicación en el depurador de EGL” en la página 291
- “Ver variables en el depurador de EGL” en la página 292

---

## Recorrer una aplicación en el depurador de EGL

Como se explica en *Depurador de EGL*, el depurador de EGL proporciona los siguientes mandatos para controlar la ejecución de un programa durante una sesión de depuración:

### **Reanudar**

Ejecuta el código hasta el siguiente punto de interrupción o hasta el final del programa.

### **Ejecutar hasta línea**

Le permite seleccionar una línea ejecutable en la vista Fuente y ejecutar el código hasta esa línea.

### **Ejecutar pasos internos**

Ejecuta la siguiente sentencia EGL y se pone en pausa. El programa se detiene en la primera sentencia de una función llamada.

### **Ejecutar pasos externos**

Ejecuta la siguiente sentencia EGL y se pone en pausa, pero no se detiene dentro de funciones que se invocan desde la función actual.

### **Recorrer hasta retorno**

Vuelve a un programa o función de invocación.

Con la excepción de Ejecutar hasta línea, puede accederse a cada uno de los mandatos de las siguientes maneras:

- Pulse en el botón adecuado de la barra de herramientas de la vista Depurar; o bien
- Pulse en el elemento de menú adecuado del menú Ejecutar; o bien
- Pulse con el botón derecho del ratón en una hebra resaltada en la vista Depurar y, a continuación, pulse en el elemento de menú adecuado.

Para utilizar Ejecutar hasta línea, haga lo siguiente cuando el programa esté en pausa:

1. Sitúe el cursor en el margen izquierdo de la vista Fuente en una línea ejecutable y, a continuación, pulse con el botón derecho del ratón. Aparecerá un menú de contexto.
2. Pulse en **Ejecutar hasta línea**.

Al utilizar Ejecutar hasta línea, tenga en cuenta lo siguiente:

- La operación no está disponible desde la vista Depurar o el menú Ejecutar
- Ejecutar hasta línea se detiene en los puntos de interrupción habilitados

### **Conceptos relacionados**

“Depurador EGL” en la página 279

### **Tareas relacionadas**

“Crear una configuración de lanzamiento en el depurador de EGL” en la página 286

“Iniciar una aplicación no de J2EE en el depurador de EGL” en la página 285

“Utilizar puntos de interrupción en el depurador de EGL” en la página 290

“Ver variables en el depurador de EGL” en la página 292

---

## Ver variables en el depurador de EGL

Siempre que un programa está en pausa, puede ver los valores actuales de las variables del programa.

Para ver las variables de un programa, haga lo siguiente:

1. En la vista Variables, expanda los componentes en el navegador para ver sus variables.
2. Para visualizar los tipos de las variables, pulse el botón **Mostrar nombres de tipo** en la barra de herramientas.
3. Para visualizar los detalles de una variable en un panel aparte, pulse en la variable y, a continuación, pulse en el botón **Mostrar detalles** de la barra de herramientas.

### Conceptos relacionados

“Depurador EGL” en la página 279

### Tareas relacionadas

“Crear una configuración de lanzamiento en el depurador de EGL” en la página 286

“Iniciar una aplicación no de J2EE en el depurador de EGL” en la página 285

“Recorrer una aplicación en el depurador de EGL” en la página 291

“Utilizar puntos de interrupción en el depurador de EGL” en la página 290

---

## Trabajar con componentes de construcción de EGL

---

### Crear un archivo de construcción

Para crear un archivo de construcción, haga lo siguiente:

1. Identifique un proyecto o carpeta para que contenga el archivo. Debe crear un proyecto o carpeta si no tiene uno todavía. El proyecto deberá un proyecto EGL o EGL Web.
2. En el entorno de trabajo, pulse en **Archivo > Nuevo > Archivo de construcción EGL**.
3. Seleccione el proyecto o carpeta que contendrá el archivo de construcción de EGL. En el campo Nombre de archivo, teclee el nombre del archivo de construcción de EGL, por ejemplo MyEGLbuildParts. Es necesaria .eglbl la extensión para el nombre de archivo. Se añade una extensión automáticamente al final del nombre de archivo si no se ha especificado una extensión o se ha especificado una extensión no válida.
4. Pulse en **Finalizar** para crear el archivo de construcción sin una declaración de componente de construcción de EGL. El archivo de construcción aparece en la vista Explorador de proyectos y se abre automáticamente en el editor de componentes de construcción de EGL por omisión.
5. Para añadir un componente de construcción de EGL antes de crear el archivo de construcción, pulse en **Siguiente**. Seleccione el tipo de componente de construcción a añadir y, a continuación, pulse en **Siguiente**. Teclee un nombre y una descripción para el componente de construcción y, a continuación, pulse en **Finalizar**. El archivo de construcción aparece en la vista Explorador de proyectos y se abre automáticamente en el editor de componentes de construcción de EGL por omisión.

#### Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13

“Introducción a EGL” en la página 1

#### Tareas relacionadas

“Añadir componente descriptor construcción a archivo construcción EGL” en la página 298

“Añadir un componente de opciones de enlace a archivo construcción EGL” en la página 313

“Añadir una sentencia de importación a un archivo de construcción EGL” en la página 319

“Añadir componente asociaciones recursos a archivo de construcción EGL” en la página 309

“Crear un proyecto Web EGL” en la página 126

## Configurar opciones de construcción generales

### Componente descriptor de construcción

Un componente descriptor de construcción controla el proceso de generación. El componente contiene varios tipos de información:

- Las *opciones del descriptor de construcción* especifican cómo generar y preparar la salida EGL, y un subconjunto de las opciones del descriptor de construcción puede hacer que se incluyan otros componentes de construcción en el proceso de generación. Para obtener información detallada sobre opciones concretas, consulte la sección *Opciones del descriptor de construcción*.

- Las *propiedades de entorno de ejecución Java* asignan valores a las siguientes propiedades:
  - `vgj.datemask.gregorian.long.entornoLocal`, que contiene la máscara de fecha utilizada en uno de estos dos casos:
    - Se invoca el código Java generado para la variable de sistema `VGVar.currentFormattedGregorianCalendarDate`; o bien
    - EGL valida un campo de formulario de texto o elemento de página que tiene una longitud de 10 o más, si la propiedad de elemento **dateFormat** está establecida en *formatoFechaGregorianoSistema*.
 El significado de *entornoLocal* se describe al final de esta sección.
  - `vgj.datemask.gregorian.short.entornoLocal`, que contiene la máscara de fecha utilizada cuando EGL valida un campo de formulario de texto o elemento de página que tiene una longitud inferior a 10, si la propiedad de elemento **dateFormat** está establecida en *formatoFechaGregorianoSistema*.
 El significado de *entornoLocal* se describe al final de esta sección.
  - `vgj.datemask.julian.long.entornoLocal`, que contiene la máscara de fecha utilizada en uno de estos dos casos:
    - Se invoca el código Java generado para la variable de sistema `VGVar.currentFormattedJulianDate`; o bien
    - EGL valida un campo de formulario de texto o elemento de página que tiene una longitud de 8 o más, si la propiedad de elemento **dateFormat** está establecida en *formatoFechaJulianoSistema*.
 El significado de *entornoLocal* se describe al final de esta sección.
  - `vgj.datemask.julian.short.entornoLocal`, que contiene la máscara de fecha utilizada cuando EGL valida un campo de formulario de texto o elemento de página que tiene una longitud inferior a 10, si la propiedad de elemento **dateFormat** está establecida en *formatoFechaJulianoSistema*.
 El significado de *entornoLocal* se describe al final de esta sección.
  - `vgj.jdbc.database.SN`, que identifica una base de datos que está disponible en el código Java.
 Debe personalizar el nombre de la propiedad cuando especifique un valor de sustitución para *SN* durante el despliegue. El valor de sustitución debe coincidir a su vez con el nombre de servidor incluido en la invocación de `VGLib.connectionService` o el nombre de base de datos incluido en la invocación de `sysLib.connect`.
 También debe personalizar el nombre de las propiedades de máscara de fecha:
    - En una unidad de ejecución determinada, cada propiedad que inicialmente está en vigor tiene un nombre cuyo último calificador (el *entornoLocal*) coincide con el valor de la propiedad de programa **vgj.nls.code**
    - En una aplicación Web, se aplica un conjunto distinto de propiedades si un programa establece la variable de sistema `sysLib.setLocale`

**Descriptor de construcción maestros:** El administrador del sistema puede requerir que utilice un *descriptor de construcción maestro* para especificar información que no puede alterarse temporalmente y que está en vigor para cada generación que se produce en la instalación de EGL. Mediante un mecanismo que se describe en *Descriptor de construcción maestro*, el administrador del sistema identifica dicho componente por nombre, junto con el archivo de construcción EGL que contiene el componente.

Si la información del descriptor de construcción maestro no es suficiente para un determinado proceso de generación o si no se identifica ningún descriptor de

construcción maestro, puede especificar un descriptor de construcción durante la generación, junto con el archivo de construcción EGL que contiene el componente específico de la generación. El descriptor de construcción específico de la generación (al igual que el descriptor de construcción maestro) debe estar en el nivel superior de un archivo de construcción EGL.

Puede crear una cadena de descriptores de construcción a partir del descriptor de construcción específico de la generación, de modo que el primero de la cadena se procesa antes que el segundo y el segundo antes que el tercero. Cuando defina un determinado descriptor de construcción, empiece una cadena (o continúe una ya empezada) asignando un valor a la opción del descriptor de construcción **nextBuildDescriptor**. El administrador del sistema puede utilizar la misma técnica para crear una cadena a partir del descriptor de construcción maestro. La implicación de encadenar la información se describe más adelante.

Cualquier componente de construcción al que hace referencia un descriptor de construcción debe estar visible en el descriptor de construcción referenciador, de acuerdo con las normas descritas en la sección Referencias a componentes. Por ejemplo, el componente de construcción puede ser un componente de opciones de enlace o un componente de asociaciones de recursos, o bien el siguiente descriptor de construcción.

**Preferencia de las opciones:** Para una determinada opción del descriptor de construcción (o propiedad de entorno de ejecución Java), el valor que se procesa inicialmente durante la generación permanece en vigor, y el orden general de preferencia es el siguiente:

1. El descriptor de construcción maestro
2. El descriptor de construcción específico de la generación, seguido de la cadena que se extiende desde el mismo
3. La cadena que se extiende desde el descriptor de construcción maestro

La ventaja de este esquema es la comodidad:

- El administrador del sistema puede especificar valores que no cambian configurando un descriptor de construcción maestro.
- Puede utilizar un descriptor de construcción específico de la generación para asignar valores que son específicos de una generación.
- Un gestor de proyectos puede especificar un conjunto de valores por omisión personalizando uno o más descriptores de construcción. En la mayoría de situaciones de este tipo, el descriptor de construcción específico de la generación señala al primer descriptor de construcción de una cadena que ha desarrollado el gestor de proyectos.

Las opciones por omisión pueden ser útiles cuando la organización desarrolla un conjunto de programas que deben generarse o prepararse de forma parecida.

- El administrador del sistema puede crear un conjunto de valores por omisión generales estableciendo una cadena que se extiende desde el descriptor de construcción maestro, aunque el uso de esta característica es poco habitual.

Si un determinado descriptor de construcción se utiliza más de una vez, sólo está en vigor el primer acceso a este descriptor de construcción. Además, sólo está en vigor la primera especificación de una determinada opción.

**Ejemplo:** Supongamos que el descriptor de construcción maestro contiene los siguientes pares opción-valor (irreales):

|         |    |
|---------|----|
| OptionX | 02 |
| OptionY | 05 |

En este ejemplo, el descriptor de construcción específico de la generación (llamado myGen) contiene los siguientes pares opción-valor.

|         |    |
|---------|----|
| OptionA | 20 |
| OptionB | 30 |
| OptionC | 40 |
| OptionX | 50 |

Como se identifica en myGen, el siguiente descriptor de construcción es myNext01, que contiene:

|         |     |
|---------|-----|
| OptionA | 120 |
| OptionD | 150 |

Como se identifica en myNext01, el siguiente descriptor de construcción es myNext02, que contiene:

|         |     |
|---------|-----|
| OptionB | 220 |
| OptionD | 260 |
| OptionE | 270 |

Como se identifica en el descriptor de construcción maestro, el siguiente descriptor de construcción es myNext99, que contiene:

|         |    |
|---------|----|
| OptionZ | 99 |
|---------|----|

EGL acepta los valores de opción en el siguiente orden:

1. Valores para las opciones del descriptor de construcción maestro:

|         |    |
|---------|----|
| OptionX | 02 |
| OptionY | 05 |

Estas opciones alteran temporalmente todas las demás.

2. Valores del descriptor de construcción específico de la generación myGen:

|         |    |
|---------|----|
| OptionA | 20 |
| OptionB | 30 |
| OptionC | 40 |

El valor para optionX en myGen no se ha tenido en cuenta.

3. Valores para las otras opciones en myNext01 y myNext02:

|         |     |
|---------|-----|
| OptionD | 150 |
| OptionE | 270 |

El valor para optionA en myNext01 no se ha tenido en cuenta, al igual que el valor para optionD en myNext02.

4. Valores para las otras opciones en myNext99:

|         |    |
|---------|----|
| OptionZ | 99 |
|---------|----|

### Conceptos relacionados

“Construcción” en la página 326

“Propiedades de tiempo de ejecución Java” en la página 347

“Referencias a componentes” en la página 21

“Descriptor de construcción maestro” en la página 297

“Componentes” en la página 17

### Tareas relacionadas

“Añadir componente descriptor construcción a archivo construcción EGL” en la página 298

“Añadir componente asociaciones recursos a archivo de construcción EGL” en la página 309

“Editar opciones generales en un descriptor de construcción” en la página 299

“Editar propiedades de entorno ejecuc. Java en descriptor construcción” en la página 302  
“Editar componente de asociaciones recursos en archivo construcción EGL” en la página 309

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382  
“Propiedades de ejecución de Java (detalles)” en la página 540

“connect()” en la página 895  
“connectionService()” en la página 916  
“setLocale()” en la página 907  
“currentFormattedGregorianCalendar” en la página 941  
“currentFormattedJulianDate” en la página 942

### Descriptor de construcción maestro

Una instalación puede proporcionar un conjunto propio de valores por omisión para las opciones de construcción y controla si estos valores por omisión pueden alterarse temporalmente.

Para configurar el descriptor de construcción maestro, cree dos componentes descriptores de construcción en el mismo archivo de construcción, donde el primero haga referencia al segundo utilizando la opción del descriptor de construcción **nextBuildDescriptor**. Las opciones del primer componente especifican valores por omisión para las opciones que no pueden alterarse temporalmente. Las opciones del segundo componente especifican valores por omisión para las opciones que pueden alterarse temporalmente.

Para instalar el descriptor de construcción maestro en el entorno de trabajo, añada un archivo xml de conector como el siguiente en el directorio de conectores del entorno de trabajo:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
 id="egl.master.build.descriptor.plugin"
 name="EGL Master Build Descriptor Plug-in"
 version="5.0"
 vendor-name="IBM">
 <requires />
 <runtime />
 <extension point =
 "com.ibm.etools.egl.generation.base.framework.masterBuildDescriptor">
 <masterBuildDescriptor
 file = "víaAccesoArchivo.nombreArchivoConstrucción"
 name = "nombreComponenteConstrucciónMaestro" />
 </extension>
</plugin>
```

La vía de acceso del archivo (*víaAccesoArchivo*) depende del directorio del espacio de trabajo.

Si utiliza el SDK de EGL, declare el nombre y el nombre vía de acceso de archivo del descriptor de construcción maestro en un archivo llamado `eglmaster.properties`. Este archivo debe estar en un directorio que se lista en la variable de entorno `CLASSPATH`. El formato del archivo de propiedades es el siguiente:

```
masterBuildDescriptorName=masterBuildPartName
masterBuildDescriptorFile=fullyQualifiedPathforEGLBuildFile
```

### Conceptos relacionados

“Construcción” en la página 326  
“Componente descriptor de construcción” en la página 293

“Plan de construcción” en la página 327

“Proyectos, paquetes y archivos EGL” en la página 13

#### **Tareas relacionadas**

“Añadir componente descriptor construcción a archivo construcción EGL”

#### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

“Formato del archivo eglmaster.properties” en la página 490

“Formato del archivo plugin.xml del descriptor de construcción maestro” en la página 506

### **Añadir componente descriptor construcción a archivo construcción EGL**

Un componente de descriptor de construcción controla el proceso de generación. Contiene nombres de opciones y sus valores relacionados, y esos pares de opción y valor especifican cómo generar y preparar la salida de EGL. Algunas opciones especifican otros componentes de control, tales como un componente de asociación de recurso, que se encuentran en el proceso de generación. Puede añadir un descriptor de construcción a un archivo de construcción de EGL. Consulte *Componente de descriptor de construcción* para obtener más información. Para añadir un componente de descriptor de construcción, haga lo siguiente:

1. Para abrir un archivo de construcción EGL con el editor de componentes de construcción EGL, haga lo siguiente en el Explorador de proyectos:
  - a. Pulse con el botón derecho del ratón en el archivo de construcción de EGL
  - b. Seleccione **Abrir con > Editor de componentes de construcción de EGL**.
2. Si la vista Esquema no se visualiza, ábrala seleccionando **Mostrar vista > Esquema** en el menú Ventana.
3. En la vista Esquema, pulse con el botón derecho del ratón en el archivo de construcción y, a continuación, pulse en **Añadir componente**.
4. Pulse en el botón de selección **Descriptor de construcción** y, a continuación, pulse en **Siguiente**.
5. Elija un nombre para el descriptor de construcción que se ajuste a los convenios de denominación de componentes de EGL. En el campo Nombre, teclee el nombre del descriptor de construcción.
6. En el campo Descripción, teclee una descripción del componente de construcción.
7. Pulse en **Finalizar**. El descriptor de construcción se declara en el archivo de construcción de EGL y las opciones generales del descriptor de construcción se visualizan en el editor de componentes de construcción de EGL.
8. Opcionalmente, puede crear una cadena de descriptores de construcción, de forma que el primero de la cadena se procese antes que el segundo y el segundo antes que el tercero. Si desea iniciar o continuar una cadena de descriptores de construcción, especifique el siguiente descriptor de construcción en el campo de opción **nextBuildDescriptor** de la lista Opciones. Para rellenar el campo de opción **nextBuildDescriptor**, haga lo siguiente:
  - a. Utilizando la barra de desplazamiento en la lista Opciones, desplácese hasta que la opción **nextBuildDescriptor** esté a la vista.
  - b. Si la fila **nextBuildDescriptor** no está resaltada, pulse en ella una vez para seleccionarla.
  - c. Pulse en el campo Valor una vez para colocar el campo en modalidad de edición.

- d. Puede teclear el nombre del siguiente descriptor de construcción en el campo Valor o seleccionar un descriptor de construcción existente en la lista desplegable.

#### Conceptos relacionados

“Componente descriptor de construcción” en la página 293

#### Tareas relacionadas

“Editar opciones generales en un descriptor de construcción”

“Editar propiedades de entorno ejecuc. Java en descriptor construcción” en la página 302

“Eliminar componente descriptor construcción de archivo construcción EGL” en la página 304

#### Consulta relacionada

“Formato de un archivo de construcción EGL” en la página 380

“Convenios de denominación” en la página 672

### Editar opciones generales en un descriptor de construcción

Un componente de descriptor de construcción controla el proceso de generación. Para editar las opciones generales del descriptor de construcción y los parámetros simbólicos, haga lo siguiente:

1. Para abrir un archivo de construcción EGL con el editor de componentes de construcción EGL, haga lo siguiente en el Explorador de proyectos:
  - a. Pulse con el botón derecho del ratón en el archivo de construcción de EGL
  - b. Seleccione **Abrir con > Editor de componentes de construcción de EGL**
2. Si la vista Esquema no se visualiza, ábrala seleccionando **Mostrar vista > Esquema** en el menú Ventana.
3. En la vista Esquema, pulse con el botón derecho del ratón en un descriptor de construcción y seleccione **Abrir**. Hay dos botones en la esquina superior derecha de la vista del editor. Asegúrese de que el botón **Mostrar opciones generales del descriptor de construcción** (el primero de los dos botones) está pulsado. El editor de componentes de construcción de EGL visualiza las opciones generales del descriptor de construcción para la definición de componente actual.
4. Opcionalmente, puede crear una cadena de descriptores de construcción, de forma que el primero de la cadena se procese antes que el segundo y el segundo antes que el tercero. Si desea iniciar o continuar una cadena de descriptores de construcción, especifique el siguiente descriptor de construcción en el campo **nextBuildDescriptor**. Si la fila **nextBuildDescriptor** no está resaltada, pulse en ella una vez para seleccionarla y, a continuación, pulse una vez en el campo Valor para poner el campo en modalidad de edición. Puede teclear el nombre del siguiente descriptor de construcción en el campo Valor o seleccionar un descriptor de construcción existente en la lista desplegable.
5. Para especificar la generación y preparación de la salida de EGL, seleccione una agrupación de pares de opción y valor de la lista desplegable **Filtro de opciones de versión de producto**. Si la opción que desea definir no está resaltada, pulse una vez para seleccionar la fila y a continuación, pulse una vez en el campo Valor para poner el campo en modalidad de edición. Puede teclear el valor de opción, o si hay una lista desplegable disponible, seleccione un valor existente. Si desea limitar la vista de pares de opción y valor a los que haya definido, pulse en el recuadro de selección **Mostrar sólo opciones especificadas**.

#### Conceptos relacionados

“Componente descriptor de construcción” en la página 293

### **Tareas relacionadas**

“Añadir componente descriptor construcción a archivo construcción EGL” en la página 298

“Editar propiedades de entorno ejecuc. Java en descriptor construcción” en la página 302

“Eliminar componente descriptor construcción de archivo construcción EGL” en la página 304

### **Consulta relacionada**

“Formato de un archivo de construcción EGL” en la página 380

## **Elegir opciones para la generación de Java**

Las opciones del descriptor de construcción se establecen en componentes del descriptor de construcción. Para elegir opciones del descriptor de construcción, inicie el editor de EGL y edite el componente de descriptor de construcción.

Al empezar a editar un componente de descriptor de construcción desde la GUI, el editor de EGL contiene un panel que lista todas las opciones del descriptor de construcción de EGL. Para limitar la visualización a las opciones aplicables a un programa, seleccione una categoría en el menú desplegable de filtros de opciones de construcción.

Seleccione cada opción que desee y establezca su valor. El valor puede ser literal, simbólico o una combinación de literal y simbólico. Puede definir parámetros simbólicos en el editor de componentes de EGL; encontrará los detalles en *Editar opciones generales del descriptor de construcción*.

Dos opciones del descriptor de construcción, **genDirectory** y **destDirectory**, le permiten utilizar un parámetro simbólico para el valor o una parte del valor. Por ejemplo, para el valor de **genDirectory** puede especificar `C:\genout\%EZEENV%`. Entonces, si genera para un entorno Windows, el directorio de generación será `C:\genout\WIN`.

No es necesario especificar todas las opciones listadas. Si no especifica un valor para una opción del descriptor de construcción, se utiliza el valor por omisión para la opción cuando la opción es aplicable en el contexto de generación.

Si ha especificado un descriptor de construcción maestro, los valores de opción de ese descriptor de construcción alteran temporalmente los valores de todos los demás descriptores de construcción. Al generar, los descriptores de construcción maestro y de generación pueden encadenarse a otros descriptores de construcción, tal como se describe en *Componente descriptor de construcción*.

### **Conceptos relacionados**

“Componente descriptor de construcción” en la página 293

### **Tareas relacionadas**

“Editar opciones generales en un descriptor de construcción” en la página 299

“Generar envolturas Java”

### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

## **Generar envolturas Java**

Puede generar clases de envolturas Java al generar el programa relacionado. Encontrará los detalles sobre cómo configurar el descriptor de construcción en *Envoltura Java*.

### Conceptos relacionados

“Generación” en la página 323

“Generación de código Java en un proyecto” en la página 323 “Envoltura Java”

### Tareas relacionadas

“Construir la salida de EGL” en la página 327

“Procesar código Java generado en un directorio” en la página 335

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“Clases de envoltura Java” en la página 551

“Salida de la generación de envoltura Java” en la página 676

**Envoltura Java:** Una envoltura Java es un conjunto de clases que actúan como interfaz entre los siguientes ejecutables:

- Por un lado, un servlet o un programa Java escrito a mano
- Y por otro lado, un bean de sesión EJB o programa generado

Las clases de envoltura Java se generan si utiliza un descriptor de construcción que tiene las siguientes características:

- La opción del descriptor de construcción **enableJavaWrapperGen** está establecida en **yes** u **only**; y
- La opción del descriptor de construcción **linkage** hace referencia a un componente de opciones de enlace que incluye un elemento **callLink** para controlar la llamada desde la envoltura al programa; y
- Se cumple una de las dos afirmaciones siguientes:
  - La llamada desde la envoltura al programa se realiza mediante un bean de sesión EJB (en cuyo caso el elemento **callLink**, propiedad **linkType** se establece en **ejbCall**); o bien
  - La llamada desde la envoltura al programa es remota (en cuyo caso, el elemento **callLink**, propiedad **type** se establece en **remoteCall**); además, el elemento **callLink**, propiedad **javaWrapper** se establece en **yes**.

Si un bean de sesión EJB hace de intermediario entre las clases de envoltura Java y un programa generado por EGL, genere la sesión EJB si utiliza un descriptor de construcción que tiene las siguientes características:

- La opción del descriptor de construcción **enableJavaWrapperGen** está establecida en **yes** u **only**; y
- La opción del descriptor de construcción **linkage** hace referencia a un componente de opciones de enlace que incluye un elemento **callLink** para controlar la llamada desde la envoltura al bean de sesión EJB (en cuyo caso, la propiedad **type** del elemento **callLink** se establece en **ejbCall**).

Para obtener más información sobre cómo utilizar las clases, consulte la sección *Clases de envoltura Java*. Para obtener información detallada sobre los nombres de clase, consulte la sección *Salida generada (referencia)*.

### Conceptos relacionados

“Salida generada” en la página 529

“Programa Java, PageHandler y biblioteca” en la página 328

“Configuraciones de tiempo de ejecución” en la página 9

### Tareas relacionadas

“Generar envolturas Java” en la página 300

### Consulta relacionada

“Salida generada (referencia)” en la página 530

“Clases de envoltura Java” en la página 551

“Salida de la generación de envoltura Java” en la página 676

## Editar propiedades de entorno ejecuc. Java en descriptor construcción

Al editar un componente de descriptor de construcción, puede asignar valores a las siguientes propiedades del entorno de ejecución Java, detalladas en *Propiedades de entorno de ejecución Java (detalles)*:

- `vgj.jdbc.database.SN`
- `vgj.datemask.gregorian.long.entorno local`
- `vgj.datemask.gregorian.short.entorno local`
- `vgj.datemask.julian.long.entorno local`
- `vgj.datemask.julian.short.entorno local`

Las asignaciones que realice se utilizan solamente si genera código Java.

Para editar las propiedades, haga lo siguiente:

1. Para abrir un archivo de construcción EGL con el editor de componentes de construcción EGL, haga lo siguiente en el Explorador de proyectos:
  - a. Pulse con el botón derecho del ratón en el archivo de construcción de EGL
  - b. Seleccione **Abrir con > Editor de componentes de construcción de EGL**
2. Si la vista Esquema no se visualiza, ábrala seleccionando **Mostrar vista > Esquema** en el menú Ventana.
3. En la vista Esquema, pulse con el botón derecho del ratón en un descriptor de construcción y seleccione **Abrir**. El editor de componentes de EGL visualiza las opciones generales del descriptor de construcción para la definición de componente actual.
4. Pulse en el botón **Mostrar propiedades de entorno de ejecución Java** en la barra de herramientas del editor.
5. Para añadir la propiedad de entorno de ejecución Java `vgj.jdbc.database.SN`, haga lo siguiente:
  - a. En el área de la pantalla titulada “Correlaciones de bases de datos para conexión”, pulse en el botón **Añadir**
  - b. Teclee un “Nombre de servidor” que utilice al codificar la palabra del sistema `VGLib.connectionService`; este valor se sustituye por `SN` en el nombre de la propiedad generada
  - c. Si la fila en la lista Correlaciones de bases de datos para conexión no está resaltada, pulse en ella una vez para seleccionarla y, a continuación, pulse una vez en el nombre JNDI o el campo de URL para poner el campo en modalidad de edición. Teclee un valor cuyo significado sea distinto para las conexiones J2EE en comparación con las conexiones no J2EE:
    - En relación con las conexiones J2EE (necesario en un entorno de producción), el valor es el nombre al que se enlaza el origen de datos en el registro de JNDI; por ejemplo, `jdbc/MyDB`
    - En relación con una conexión JDBC estándar (como la que podría utilizarse para depurar), el valor es la URL de conexión; por ejemplo, `jdbc:db2:MyDB`

6. Para asignar las máscaras de fecha utilizadas al codificar `VGVar.currentFormattedGregorianDate` (para una fecha Gregoriana) o `VGVar.currentFormattedJulianDate` (para una fecha Juliana); o EGL valida un elemento de página o un campo de formulario de texto que tenga una longitud de 10 o más y una propiedad **dateFormat** de *formatoFechaGregorianoSistema* o *FormatoFechaJulianoSistema*, haga lo siguiente:
  - a. En el área de la pantalla titulada "Máscaras de fecha", pulse en el botón **Añadir**
  - b. En la columna Entorno local, seleccione uno de los códigos del cuadro de lista; el valor seleccionado se sustituye por *entorno local* en las propiedades de máscara de fecha listadas anteriormente. Solamente se utiliza una de las entradas durante la ejecución: la entrada para la cual el valor de *entorno local* coincide con el valor de la propiedad de entorno de ejecución Java `vgj.nls.code`
  - c. Si la fila en la lista Máscaras de fecha no está resaltada, pulse en ella una vez para seleccionarla y, a continuación, pulse una vez en el campo Máscara Gregoriana larga para poner el campo en modalidad de edición. Seleccione una máscara en el cuadro de lista o bien teclee una máscara; los caracteres que no sean D, Y o dígitos pueden utilizarse como separadores, y el valor por omisión es específico del entorno local
  - d. Si la fila en la lista Máscaras de fecha no está resaltada, pulse en ella una vez para seleccionarla y, a continuación, pulse una vez en el campo Máscara Juliana larga para poner el campo en modalidad de edición. Seleccione una máscara en el cuadro de lista o bien teclee una máscara; los caracteres que no sean D, Y o dígitos pueden utilizarse como separadores, y el valor por omisión es específico del entorno local
7. Para asignar las máscaras de fecha utilizadas cuando EGL valida un elemento de página o un campo de formulario de texto que tenga una longitud de menos de 10 y una propiedad **dateFormat** de *formatoFechaGregorianoSistema* o *formatoFechaJulianoSistema*, haga lo siguiente:
  - a. En el área de la pantalla titulada "Máscaras de fecha", pulse en el botón **Añadir**
  - b. En la columna Entorno local, seleccione uno de los códigos del cuadro de lista; el valor seleccionado se sustituye por *entorno local* en las propiedades de máscara de fecha listadas anteriormente. Solamente se utiliza una de las entradas durante la ejecución: la entrada para la cual el valor de *entorno local* coincide con el valor de la propiedad de entorno de ejecución Java `vgj.nls.code`
  - c. Si la fila en la lista Máscaras de fecha no está resaltada, pulse en ella una vez para seleccionarla y, a continuación, pulse una vez en el campo Máscara Gregoriana corta para poner el campo en modalidad de edición. Seleccione una máscara en el cuadro de lista o bien teclee una máscara; los caracteres que no sean D, Y o dígitos pueden utilizarse como separadores, y el valor por omisión es específico del entorno local
  - d. Si la fila en la lista Máscaras de fecha no está resaltada, pulse en ella una vez para seleccionarla y, a continuación, pulse una vez en el campo Máscara Juliana corta para poner el campo en modalidad de edición. Seleccione una máscara en el cuadro de lista o bien teclee una máscara; los caracteres que no sean D, Y o dígitos pueden utilizarse como separadores, y el valor por omisión es específico del entorno local
8. Para eliminar una asignación, pulse en ella y, a continuación, pulse en el botón **Eliminar**.

### Conceptos relacionados

“Componente descriptor de construcción” en la página 293

“Propiedades de tiempo de ejecución Java” en la página 347

### Tareas relacionadas

“Añadir componente descriptor construcción a archivo construcción EGL” en la página 298

“Editar opciones generales en un descriptor de construcción” en la página 299

“Eliminar componente descriptor construcción de archivo construcción EGL”

### Consulta relacionada

“Formato de un archivo de construcción EGL” en la página 380

“Propiedades de ejecución de Java (detalles)” en la página 540

“connectionService()” en la página 916

“currentFormattedGregorianCalendar” en la página 941

“currentFormattedJulianDate” en la página 942

## Eliminar componente descriptor construcción de archivo construcción EGL

Para eliminar un componente de descriptor de construcción de un archivo de construcción de EGL, haga lo siguiente:

1. Para abrir un archivo de construcción EGL con el editor de componentes de construcción EGL, haga lo siguiente en el Explorador de proyectos:
  - a. Pulse con el botón derecho del ratón en el archivo de construcción de EGL
  - b. Seleccione **Abrir con > Editor de componentes de construcción de EGL**
2. Si la vista Esquema no se visualiza, ábrala seleccionando **Mostrar vista > Esquema** en el menú Ventana
3. En la vista Esquema, pulse con el botón derecho del ratón sobre el componente de descriptor de construcción y pulse **Eliminar**

### Conceptos relacionados

“Componente descriptor de construcción” en la página 293

### Tareas relacionadas

“Añadir componente descriptor construcción a archivo construcción EGL” en la página 298

“Editar opciones generales en un descriptor de construcción” en la página 299

“Editar propiedades de entorno ejecuc. Java en descriptor construcción” en la página 302

## Configurar asociaciones de archivo externo, impresora y colas

### Asociaciones de recursos y tipos de archivo

Un registro fijo EGL que accede a una cola, impresora o archivo externo tiene un nombre de cola o archivo lógico. (En el caso de una impresora, el nombre de archivo lógico es *impresora* para la mayoría de sistemas de ejecución.) El nombre no puede tener más de 8 caracteres y sólo es significativo como una forma de relacionar el registro con un *nombre de sistema*, que el sistema destino utiliza para acceder a una cola, impresora o archivo físico.

En relación con los archivos o colas, el nombre de archivo o de cola es un valor por omisión para el nombre de sistema. En relación con las impresoras, no existe ningún valor por omisión.

En lugar de aceptar un valor por omisión, puede realizar una o las dos acciones siguientes:

- Durante la generación, controle el proceso de generación con un descriptor de construcción que, a su vez, hace referencia a un determinado componente de asociaciones de recursos. El componente de asociaciones de recursos relaciona el nombre de archivo con un nombre de sistema de la plataforma destino donde tiene previsto desplegar el código generado.
- Durante la ejecución (en la mayoría de casos) puede cambiar el valor en la variable específica del registro `resourceAssociation` (para archivos o colas) o en la variable de sistema `ConverseVar.printerAssociation` (para la salida de impresión). El objetivo es alterar temporalmente el nombre de sistema que ha establecido ya sea por omisión o bien especificando un componente de asociaciones de recursos.

El componente de asociaciones de recursos no se aplica a los siguientes tipos de registros:

- `basicRecord`, ya que los registros básicos no interactúan con almacenes de datos
- `SQLRecord`, ya que los registros SQL interactúan con bases de datos relacionales

**Componente de asociaciones de recursos:** El componente de asociaciones de recursos es un conjunto de elementos de asociaciones, cada uno de los cuales tiene las siguientes características:

- Es específico de un nombre de cola o archivo lógico
- Tiene un conjunto de entradas, cada una de las cuales es específica de un sistema destino; cada entrada identifica el tipo de archivo de la plataforma destino, junto con el nombre de sistema y en algunos casos información adicional

Un elemento de asociaciones puede interpretarse como un conjunto de propiedades y valores en una relación jerárquica, como en el siguiente ejemplo:

```
// un elemento de asociaciones
property: fileName
value: myFile01

// una entrada, con varias propiedades
property: system
value: aix
property: fileType
value: spool
property: systemName
value: employee

// una segunda entrada
property: system
value: win
property: fileType
value: seqws
property: systemName
value: c:\myProduct\myFile.txt
```

En este ejemplo, el nombre de archivo `myFile01` está relacionado con los siguientes archivos:

- *employee* en AIX
- *myFile.txt* en Windows 2000/NT/XP

El nombre de archivo debe ser un nombre válido, un asterisco o el principio de un nombre válido seguido de un asterisco. El asterisco es el equivalente del comodín de uno o más caracteres y proporciona una forma de identificar un conjunto de

nombres. Por ejemplo, un elemento de asociaciones que incluye el siguiente valor para un nombre de archivo hace referencia a cualquier archivo que empieza por las letras *myFile*:

```
myFile*
```

Si varios elementos son válidos para un nombre de archivo que se utiliza en el programa, EGL utiliza el primer elemento que se aplica. Una serie de elementos de asociaciones, por ejemplo, podría caracterizarse por los siguientes valores para el nombre de archivo, en orden:

```
myFile
myFile*
*
```

Considere el elemento asociado al último valor, donde el valor de *myFile* sólo es un asterisco. Este elemento podría aplicarse a cualquier archivo; pero en relación con un determinado archivo, el último elemento sólo se aplica si los elementos anteriores no lo hacen. Si, por ejemplo, el programa hace referencia a *myFile01*, el enlace especificado en el segundo elemento reemplaza al tercer elemento para definir cómo se maneja la referencia.

Durante la generación, EGL selecciona un elemento de asociaciones concreto, junto con la primera entrada que sea apropiada. Una entrada es apropiada en uno de estos dos casos:

- Existe una coincidencia entre el sistema destino para el que está generando, por un lado, y la propiedad **system**, por el otro; o bien
- La propiedad **system** tiene el siguiente valor:

```
any
```

Si, por ejemplo, está generando para AIX, EGL utiliza la primera entrada que hace referencia a **aix** o a **any**.

**Tipos de archivo:** Un tipo de archivo determina las propiedades que son necesarias para una determinada entrada en un elemento de asociaciones. La tabla siguiente describe los tipos de archivo EGL.

| Tipo de archivo | Descripción                                                                                                                                                                                                                |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ibmcobol        | Un archivo VSAM al que se accede remotamente mediante un programa Java generado por EGL. Para obtener información detallada sobre cómo especificar el nombre de sistema en este caso, consulte la sección Soporte de VSAM. |
| mq              | Una cola de mensajes MQSeries; para obtener información detallada sobre cómo trabajar con una cola de este tipo, consulte la sección Soporte de MQSeries.                                                                  |
| seqws           | Un archivo serie al que se accede mediante un programa Java generado por EGL.                                                                                                                                              |
| spool           | Un archivo de spool en AIX o Linux.                                                                                                                                                                                        |

**Tipos de registros y VSAM:** Cada uno de los tres tipos de registros fijos es apropiado para acceder a un conjunto de datos VSAM, pero sólo si el tipo de archivo en el elemento de asociaciones del registro es **ibmcobol**, **vsam** o **vsamrs**:

- Si el registro fijo es de tipo **indexedRecord**, el conjunto de datos VSAM es un conjunto de datos con secuencia de clave con un índice primario o alternativo

- Si el registro fijo es de tipo `relativeRecord`, el conjunto de datos VSAM es un conjunto de datos de registro relativo
- Si el registro fijo es de tipo `serialRecord`, el conjunto de datos VSAM es un conjunto de datos con secuencia de entrada

**Para obtener más detalles:** Para obtener más detalles sobre las asociaciones de recursos, consulte los temas siguientes:

- *Referencia cruzada de tipos de archivos y registros*
- *Elementos de asociaciones*

### Conceptos relacionados

- “Componentes de registro fijo” en la página 133
- “Soporte de MQSeries” en la página 265
- “Componentes” en la página 17
- “Tipos de registros y propiedades” en la página 135
- “Componentes de registro” en la página 132
- “Soporte de VSAM” en la página 264

### Tarea relacionada

- “Añadir componente asociaciones recursos a archivo de construcción EGL” en la página 309
- “Editar componente de asociaciones recursos en archivo construcción EGL” en la página 309
- “Eliminar componente asociaciones recursos de archivo construcción EGL” en la página 311

### Consulta relacionada

- “Elementos de asociación” en la página 375
- “Referencias cruzadas de tipo de registro y tipo de archivo” en la página 737
- “recordName.resourceAssociation” en la página 859
- “resourceAssociations” en la página 398
- “system” en la página 404
- “printerAssociation” en la página 923

## Unidad lógica de trabajo

Cuando se cambian recursos que están clasificados como *no recuperables* (como por ejemplo, archivos serie en Windows 2000), el trabajo es relativamente permanente; ni el código ni los servicios de ejecución EGL pueden rescindir fácilmente los cambios. Cuando se cambian recursos que están clasificados como *recuperables* (como por ejemplo, bases de datos relacionales), el código o los servicios de ejecución EGL pueden comprometer los cambios para hacer que el trabajo sea permanente o bien pueden retrotraer los cambios para volver al contenido que estaba en vigor cuando los cambios se comprometieron por última vez.

Los recursos recuperables son los siguientes:

- Bases de datos relacionales
- Colas y archivos CICS que están configurados para ser recuperables
- Colas de mensajes MQSeries, a menos que el registro MQSeries especifique lo contrario, como se describe en la sección *Soporte de MQSeries*

Una *unidad lógica de trabajo* identifica operaciones de entrada que se comprometen o retrotraen como un grupo. Una unidad de trabajo empieza cuando el código cambia un recurso recuperable; y finaliza cuando se produce el primero de los siguientes eventos:

- El código invoca la función de sistema `sysLib.commit` o `sysLib.rollback` para comprometer o retrotraer los cambios

- Los servicios de ejecución EGL realizan una retrotracción como respuesta a un error grave que no se maneja en el código; en este caso, todos los programas de la unidad de ejecución se eliminan de la memoria
- Se produce un compromiso implícito, tal como sucede en los casos siguientes:
  - Un programa emite una sentencia **show**.
  - El programa de nivel superior de una unidad de ejecución finaliza satisfactoriamente, tal como se describe en *Unidad de ejecución*.
  - Se visualiza una página Web, al igual que cuando un PageHandler emite una sentencia **forward**.
  - Un programa emite una sentencia **converse** y cualquiera de las siguientes opciones es aplicable:
    - No está en modalidad de compatibilidad de VisualAge Generator y el programa es un programa segmentado
    - **ConverseVar.commitOnConverse** está establecido en 1
    - Está en modalidad de compatibilidad de VisualAge Generator y **ConverseVar.segmentedMode** está establecido en 1

**Unidad de trabajo para Java:** En una unidad de ejecución Java, los detalles son los siguientes:

- Cuando alguno de los programas Java finaliza con un error grave, el efecto es equivalente a realizar retrotracciones, cerrar cursores y liberar bloqueos.
- Cuando la unidad de ejecución finaliza satisfactoriamente, EGL realiza un compromiso, cierra cursores y libera bloqueos.
- Puede utilizar varias conexiones para leer varias bases de datos, pero sólo debe actualizar una base de datos en una unidad de trabajo ya que sólo está disponible un compromiso de una fase. Para obtener información relacionada, consulte la sección *VGLib.connectionService*.
- Cuando se accede a un programa generado por EGL mediante un bean de sesión EJB generado por EGL, el control de transacción puede verse afectado por un atributo de transacción (también llamado tipo de transacción de contenedor), que se encuentra en el descriptor de despliegue del bean de sesión EJB. El atributo de transacción sólo afecta al control de transacción cuando el componente de opciones de enlace, elemento callLink, propiedad **remoteComType** de la llamada es directo, como se describe en la sección *remoteComType en elemento callLink*.

El bean de sesión EJB se genera con el atributo de transacción REQUIRED, pero puede cambiar el valor durante el despliegue. Para obtener información detallada sobre las implicaciones del atributo de transacción, consulte la documentación Java.

#### Conceptos relacionados

- “Soporte de MQSeries” en la página 265
- “Unidad de ejecución” en la página 742
- “Soporte de SQL” en la página 229

#### Tareas relacionadas

- “Establecer una conexión JDBC J2EE” en la página 362
- “Cómo se realiza una conexión JDBC estándar” en la página 263

#### Consulta relacionada

- “Base de datos por omisión” en la página 251
- “commit()” en la página 893
- “connectionService()” en la página 916

“rollback()” en la página 905

“Clases de envoltura Java” en la página 551

“Propiedad luvControl del elemento callLink” en la página 414

“Propiedad remoteComType del elemento callLink” en la página 419

“sqlDB” en la página 400

## **Añadir componente asociaciones recursos a archivo de construcción EGL**

Un componente de asociaciones de recursos relaciona un nombre de archivo con un nombre de recurso del sistema en la plataforma destino en la que tiene intención de desplegar el código generado. Puede añadir un componente de asociaciones de recursos a un archivo de construcción de EGL. Encontrará los detalles en *Asociaciones de recursos y tipos de archivo*. Para añadir un componente de asociaciones de recursos, haga lo siguiente:

1. Para abrir un archivo de construcción EGL con el editor de componentes de construcción EGL, haga lo siguiente en el Explorador de proyectos:
  - a. Pulse con el botón derecho del ratón en el archivo de construcción de EGL
  - b. Seleccione **Abrir con > Editor de componentes de construcción de EGL**
2. Si la vista Esquema no se visualiza, ábrala seleccionando **Mostrar vista > Esquema** en el menú Ventana.
3. En la vista Esquema, pulse con el botón derecho del ratón en el archivo de construcción y, a continuación, pulse en **Añadir componente**.
4. Pulse en **Asociaciones de recursos** y, a continuación, pulse en **Siguiente**.
5. Elija un nombre para el componente de asociaciones de recursos que se ajuste a los convenios de denominación de componentes de EGL. En el campo Nombre, teclee el nombre del componente de asociaciones de recursos.
6. En el campo Descripción, teclee una descripción del componente.
7. Pulse en **Finalizar**. El componente de asociaciones de recursos se añade al archivo de construcción de EGL y se abre la página de componente de asociaciones de recursos en el editor de componentes de construcción de EGL.

### **Conceptos relacionados**

“Componente descriptor de construcción” en la página 293

“Asociaciones de recursos y tipos de archivo” en la página 304

### **Tareas relacionadas**

“Editar componente de asociaciones recursos en archivo construcción EGL”

“Eliminar componente asociaciones recursos de archivo construcción EGL” en la página 311

### **Consulta relacionada**

“Formato de un archivo de construcción EGL” en la página 380

“Convenios de denominación” en la página 672

## **Editar componente de asociaciones recursos en archivo construcción EGL**

Un componente de asociaciones de recursos relaciona un nombre de archivo con un nombre de recurso del sistema en la plataforma destino en la que tiene intención de desplegar el código generado.

Para editar un componente de asociaciones de recursos, haga lo siguiente:

1. Para abrir un archivo de construcción EGL con el editor de componentes de construcción EGL, haga lo siguiente en el Explorador de proyectos:
  - a. Pulse con el botón derecho del ratón en el archivo de construcción de EGL

- b. Seleccione **Abrir con > Editor de componentes de construcción de EGL**
2. Si la vista Esquema no se visualiza, ábrala seleccionando **Mostrar vista > Esquema** en el menú Ventana.
3. En la vista Esquema, pulse con el botón derecho del ratón en un componente de asociaciones de recursos y pulse en **Abrir**. El editor visualiza la definición de componente actual.
4. Para añadir un nuevo Elemento de asociación al componente, pulse en **Añadir asociación** o pulse la tecla Insertar y teclee el nombre de archivo lógico o seleccione un nombre de archivo lógico.
5. Para cambiar el nombre del sistema por omisión asociado con el nombre de archivo lógico, puede realizar una de estas acciones:
  - Seleccione la fila correspondiente en la lista de Elementos de asociación y, a continuación, pulse una vez en el nombre para poner el campo en modalidad de edición. Seleccione el nuevo nombre de sistema en la lista desplegable de Sistema.
  - En la lista de Propiedades de entradas de sistema seleccionadas, pulse una vez en la propiedad del sistema para poner el campo de Valor asociado con esa propiedad en modalidad de edición. Seleccione el nuevo nombre de sistema en la lista desplegable de Valor.
6. Para cambiar el tipo de archivo por omisión asociado con el nombre de archivo lógico, puede realizar una de estas acciones:
  - Seleccione la fila en la lista de Elementos de asociación que corresponda al nombre de archivo lógico y, a continuación, pulse una vez en el nombre para poner el campo en modalidad de edición. Seleccione el nuevo tipo de archivo en la lista desplegable de Tipo de archivo.
  - Seleccione la fila en la lista de Elementos de asociación que corresponda al nombre de archivo lógico. En la lista de Propiedades de entradas de sistema seleccionadas, pulse una vez en la propiedad fileType para poner el campo de Valor asociado con esa propiedad en modalidad de edición. Seleccione el tipo de archivo en la lista desplegable de Valor.
7. Modifique las asociaciones de recursos como sea necesario.
  - Para asociar más de un sistema y conjunto de propiedades relacionadas con un nombre de archivo lógico, seleccione la fila en la lista de Elementos de asociación que corresponda al nombre de archivo lógico. Al final de la lista Elementos de asociación, pulse en **Añadir sistema**. La fila añadida queda seleccionada y disponible para editarla.
  - Para eliminar un sistema y las propiedades relacionadas de nombre de archivo lógico asociado, seleccione la fila en la lista de Elementos de asociación que corresponda al nombre de archivo lógico. Al final de la lista Elementos de asociación, pulse en **Eliminar** o pulse la tecla Suprimir.
  - Para eliminar un nombre de archivo lógico y los sistemas asociados, seleccione la fila en la lista de Elementos de asociación que corresponda al nombre de archivo lógico. Al final de la lista Elementos de asociación, pulse en **Eliminar** o pulse la tecla Suprimir.

#### **Conceptos relacionados**

“Componente descriptor de construcción” en la página 293

“Asociaciones de recursos y tipos de archivo” en la página 304

#### **Tareas relacionadas**

“Añadir componente asociaciones recursos a archivo de construcción EGL” en la página 309

“Eliminar componente asociaciones recursos de archivo construcción EGL” en la página 311

### Consulta relacionada

“Formato de un archivo de construcción EGL” en la página 380

## Eliminar componente asociaciones recursos de archivo construcción EGL

Para eliminar un componente de asociaciones de recursos de un archivo de construcción de EGL, haga lo siguiente:

1. Para abrir un archivo de construcción EGL con el editor de componentes de construcción EGL, haga lo siguiente en el Explorador de proyectos:
  - a. Pulse con el botón derecho del ratón en el archivo de construcción de EGL
  - b. Seleccione **Abrir con > Editor de componentes de construcción de EGL**
2. Si la vista Esquema no se visualiza, ábrala seleccionando **Mostrar vista > Esquema** en el menú Ventana
3. En la vista Esquema, pulse con el botón derecho del ratón sobre el componente de asociaciones de recursos y pulse **Eliminar**

### Conceptos relacionados

“Asociaciones de recursos y tipos de archivo” en la página 304

### Tareas relacionadas

“Añadir componente asociaciones recursos a archivo de construcción EGL” en la página 309

“Editar componente de asociaciones recursos en archivo construcción EGL” en la página 309

## Configurar opciones de llamada y transferencia

### Componente de opciones de enlace

Un componente de *opciones de enlace* especifica detalles sobre los temas siguientes:

- Cómo una envoltura o un programa generado llama a otro código generado
- Cómo un programa generado se transfiere de forma asíncrona a otro programa generado

**Especificar cuándo las opciones de enlace son finales:** Puede elegir entre dos alternativas:

- Las opciones de enlace especificadas durante la generación se aplican durante la ejecución; o bien
- Las opciones de enlace especificadas en un archivo de propiedades de enlace durante el despliegue se aplican durante la ejecución. Aunque puede escribir manualmente este archivo, EGL lo genera en esta situación:
  - Cuando se establece la propiedad de opciones de enlace **remoteBind** en **RUNTIME**; y
  - Se genera un programa o envoltura Java con la opción del descriptor de construcción **genProperties** establecida en **GLOBAL** o **PROGRAM**.

Para obtener información detallada sobre cómo utilizar el archivo, consulte la sección Desplegar un archivo de propiedades de enlace. Para obtener información detallada sobre cómo personalizar el archivo, consulte la sección Archivo de propiedades de enlace (referencia).

**Elementos de un componente de opciones de enlace:** El componente de opciones de enlace se compone de un conjunto de elementos, cada uno de los cuales tiene un conjunto de propiedades y valores. Están disponibles los siguientes tipos de elementos:

- Un elemento **callLink** especifica los convenios de enlace que EGL utiliza para una determinada llamada.

Elemento `callLink` siempre se aplica a un programa llamado. Las siguientes relaciones están en vigor:

- Si el elemento `callLink` hace referencia al programa que se está generando, este elemento ayuda a determinar si se debe generar una envoltura Java que permite acceder al programa desde el código Java nativo; para obtener una visión general, consulte la sección *Envoltura Java*. Si se indica que la envoltura Java acceda al programa mediante el bean de sesión EJB, el elemento `callLink` también hace que se genere un bean de sesión EJB.
- Si genera un programa Java y si el elemento `callLink` hace referencia a un programa al que llama este programa, el elemento `callLink` especifica cómo se implementa la llamada; por ejemplo, si la llamada es local o remota. Si se indica que el programa Java llamante realice la llamada a través de un bean de sesión EJB, el elemento `callLink` hace que se genere un bean de sesión EJB.
- Un elemento `asynchLink` especifica cómo un programa generado se transfiere de forma asíncrona a otro programa, como ocurre cuando el programa que realiza la transferencia invoca la función de sistema `sysLib.startTransaction`.
- Un elemento `transferToProgram` especifica cómo un programa COBOL generado transfiere el control a un programa y finaliza el proceso. Este elemento no se utiliza para la salida Java y sólo es significativo para un programa principal que emite una sentencia **transfer** de tipo *transferir a programa*.
- Un elemento `transferToTransaction` especifica cómo un programa generado transfiere el control a una transacción y finaliza el proceso. Este elemento sólo es significativo para un programa principal que emite una sentencia **transfer** de tipo *transferir a transacción*. Sin embargo, el elemento no es necesario cuando el programa destino se genera con VisualAge Generator o (en ausencia de un alias) con EGL.

#### Identificación de programas o registros a los que hace referencia un elemento:

En cada elemento, una propiedad (por ejemplo, `pgmName`) identifica los programas o registros a los que hace referencia el elemento; y, a menos que se indique lo contrario, el valor de dicha propiedad puede ser un nombre válido, un asterisco o el principio de un nombre válido seguido de un asterisco. El asterisco es el equivalente del comodín de uno o más caracteres y proporciona una forma de identificar un conjunto de nombres.

Considere un elemento `callLink` que incluya el siguiente valor para la propiedad `pgmName`:

```
myProg*
```

Este elemento hace referencia a cualquier componente de programa EGL que empieza por las letras *myProg*.

Si varios elementos son válidos, EGL utiliza el primer elemento que se aplica. Una serie de elementos `callLink`, por ejemplo, podría caracterizarse por los siguientes valores `pgmName`, en orden:

```
YourProgram
YourProg*
*
```

Considere el elemento asociado al último valor, donde el valor de `pgmName` sólo es un asterisco. Este elemento podría aplicarse a cualquier programa; pero en relación con un determinado programa, el último elemento sólo se aplica si los elementos anteriores no lo hacen. Si, por ejemplo, el programa llama a `YourProgram01`, el enlace especificado en el segundo elemento (`YourProg*`) reemplaza el tercer elemento (\*) para definir cómo EGL maneja la llamada.

En la mayoría de casos, los elementos con nombres más específicos deben preceder a los que tienen nombres más generales. En el ejemplo anterior, el elemento con el asterisco está colocado apropiadamente para proporcionar las especificaciones del enlace por omisión.

#### **Conceptos relacionados**

“Envoltura Java” en la página 301

“Componentes” en la página 17

#### **Tareas relacionadas**

“Añadir un componente de opciones de enlace a archivo construcción EGL”

“Desplegar un archivo de propiedades de enlace” en la página 364

“Editar el elemento asynchLink de un componente de opciones de enlace” en la página 315

“Editar el elemento callLink de un componente de opciones de enlace” en la página 314

“Editar elementos relacionados con transfer. de comp. opciones enlace” en la página 317

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

#### **Consulta relacionada**

“Elemento asynchLink” en la página 377

“call” en la página 563

“Elemento callLink” en la página 407

“linkage” en la página 397

“Archivo de propiedades de enlace (detalles)” en la página 657

“startTransaction()” en la página 911

“transfer” en la página 646

“Elemento transferToTransaction” en la página 952

### **Añadir un componente de opciones de enlace a archivo construcción EGL**

Un componente de opciones de enlace describe cómo un programa EGL generado implementa llamadas y transferencias y cómo el programa accede a archivos. Para añadir este tipo de componente, haga lo siguiente:

1. Para abrir un archivo de construcción EGL con el editor de componentes de construcción EGL, haga lo siguiente en el Explorador de proyectos:
  - a. Pulse con el botón derecho del ratón en el archivo EGL
  - b. Seleccione **Abrir con > Editor de componentes de construcción de EGL**
2. Si la vista Esquema no se visualiza, ábrala seleccionando **Mostrar vista > Esquema** en el menú Ventana.
3. En la vista Esquema, pulse con el botón derecho del ratón en el archivo de construcción y, a continuación, pulse en **Añadir componente**.
4. Pulse en **Opciones de enlace** y, a continuación, pulse en **Siguiente**.
5. Elija un nombre para el componente de opciones de enlace que se ajuste a los convenios de denominación de componentes de EGL. En el campo Nombre, teclee el nombre del componente de opciones de enlace.
6. En el campo Descripción, teclee una descripción del componente.
7. Pulse en **Finalizar**. El componente de opciones de enlace se añade al archivo de EGL y se abre la página de componente de opciones de enlace en el editor de componentes de construcción de EGL.

### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

### Tareas relacionadas

“Editar el elemento asynchLink de un componente de opciones de enlace” en la página 315

“Editar el elemento callLink de un componente de opciones de enlace”

“Editar elementos relacionados con transfer. de comp. opciones enlace” en la página 317

“Eliminar componente opciones enlace a archivo de construcción de EGL” en la página 318

### Consulta relacionada

“Formato de un archivo de construcción EGL” en la página 380

“Convenios de denominación” en la página 672

## Editar el elemento callLink de un componente de opciones de enlace

Un componente de opciones de enlace describe cómo un programa EGL generado implementa llamadas y transferencias y cómo el programa accede a archivos. Para editar el elemento callLink del componente, haga lo siguiente:

1. Para abrir un archivo de construcción EGL con el editor de componentes de construcción EGL, haga lo siguiente en el Explorador de proyectos:
  - a. Pulse con el botón derecho del ratón en el archivo de construcción de EGL
  - b. Seleccione **Abrir con > Editor de componentes de construcción de EGL**
2. Si la vista Esquema no se visualiza, ábrala seleccionando **Mostrar vista > Esquema** en el menú Ventana.
3. En la vista Esquema, pulse con el botón derecho del ratón en un componente de opciones de enlace y pulse en **Abrir**. El editor de componentes de EGL visualiza la declaración de componente actual.
4. Pulse en el botón **Mostrar elementos CallLink** en la barra de herramientas del editor.
5. Para añadir un nuevo elemento CallLink, pulse en **Añadir** o pulse la tecla Insertar y teclee el Nombre de programa (pgmName) o seleccione un nombre de programa de la lista desplegable Nombre de programa.
6. Para cambiar el tipo de llamada por omisión asociado con el nombre de programa, puede realizar una de estas acciones:
  - Seleccione la fila correspondiente en la lista de elementos CallLink y, a continuación, pulse una vez en el campo Tipo (localCall, remoteCall, ejbCall) para poner el campo en modalidad de edición. Seleccione el nuevo tipo de llamada en la lista desplegable de Tipo.
  - En la lista de Propiedades de elementos callLink seleccionados, pulse una vez en la propiedad de tipo para poner el campo de Valor asociado con esa propiedad en modalidad de edición. Seleccione el nuevo tipo de llamada en la lista desplegable de Valor.
7. Otras propiedades asociadas con el nombre de programa están listadas en la lista Propiedades de elementos callLink seleccionados basada en el tipo de llamada. Para cambiar el valor de estas propiedades, seleccione el nombre de programa. En la lista de Propiedades de elementos callLink seleccionados, pulse una vez en la propiedad que desee definir para poner el campo de Valor asociado con esa propiedad en modalidad de edición. Defina el nuevo valor seleccionando una opción en la lista desplegable de Valor, o tecleando el nuevo valor en el campo Valor. Para algunas propiedades solamente puede seleccionar una opción en una lista desplegable. Para otras propiedades solamente puede teclear un valor en el campo Valor.

8. Modifique la lista de elementos CallLink como sea necesario:
- Para volver a situar un elemento callLink, seleccione un elemento y pulse **Colocar antes** o **Colocar después**.
  - Para eliminar un elemento callLink, seleccione el elemento y pulse en **Eliminar** o pulse la tecla Suprimir.

#### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

#### Tareas relacionadas

“Añadir un componente de opciones de enlace a archivo construcción EGL” en la página 313

“Editar el elemento asynchLink de un componente de opciones de enlace”

“Editar elementos relacionados con transfer. de comp. opciones enlace” en la página 317

“Eliminar componente opciones enlace a archivo de construcción de EGL” en la página 318

#### Consulta relacionada

“Elemento asynchLink” en la página 377

“Elemento callLink” en la página 407

“Formato de un archivo de construcción EGL” en la página 380

“Archivo de propiedades de enlace (detalles)” en la página 657

**Bean de sesión EJB (Enterprise JavaBean):** Un bean de sesión EJB contiene por los siguientes componentes:

- Interfaz inicial, que ofrece a un cliente acceso al bean de sesión EJB durante la ejecución
- Interfaz de bean remota, que lista los métodos que están disponibles directamente para dicho cliente
- Implementación de bean, que contiene la lógica que está disponible indirectamente para dicho cliente

Un bean de sesión EJB es un intermediario entre un programa y otro o entre una envoltura Java EGL y un programa. La generación del bean de sesión EJB depende en gran medida de los valores que tenga el componente de opciones de enlace utilizado durante la generación. Para obtener información detallada, consulte la sección *Componente de opciones de enlace*; concretamente, la visión general del elemento **callLink**.

Para obtener información detallada sobre los nombres de archivos de salida, consulte la sección *Salida generada (referencia)*.

#### Conceptos relacionados

“Salida generada” en la página 529

“Componente de opciones de enlace” en la página 311

#### Consulta relacionada

“Salida generada (referencia)” en la página 530

### Editar el elemento asynchLink de un componente de opciones de enlace

Un componente de opciones de enlace describe cómo un programa EGL generado implementa llamadas y transferencias y cómo accede a archivos. Para editar el elemento asynchLink del componente, haga lo siguiente:

1. Para abrir un archivo de construcción EGL con el editor de componentes de construcción EGL, haga lo siguiente en el Explorador de proyectos:
  - a. Pulse con el botón derecho del ratón en el archivo de construcción de EGL
  - b. Seleccione **Abrir con > Editor de componentes de construcción de EGL**
2. Si la vista Esquema no se visualiza, ábrala seleccionando **Mostrar vista > Esquema** en el menú Ventana.
3. En la vista Esquema, pulse con el botón derecho del ratón en un componente de opciones de enlace y pulse en **Abrir**. El editor de componentes de construcción de EGL visualiza la declaración de componente actual.
4. Pulse en el botón Mostrar elementos AsyncLink en la barra de herramientas del editor.
5. Para añadir un nuevo elemento AsyncLink, pulse en **Añadir** o pulse la tecla Insertar y teclee el Nombre de registro (recordName) o seleccione un nombre de registro de la lista desplegable Nombre de registro.
6. Para cambiar el tipo de enlace por omisión asociado con el nombre de registro, puede realizar una de estas acciones:
  - Seleccione la fila correspondiente en la lista de elementos AsyncLink y, a continuación, pulse una vez en el campo Tipo (localAsync, remoteAsync) para poner el campo en modalidad de edición. Seleccione el nuevo tipo de enlace en la lista desplegable de Tipo.
  - En la lista de Propiedades de elementos asyncLink seleccionados, pulse una vez en la propiedad de tipo para poner el campo de Valor asociado con esa propiedad en modalidad de edición. Seleccione el nuevo tipo de enlace en la lista desplegable de Valor.
7. Otras propiedades asociadas con el nombre de registro están listadas en la lista Propiedades de elementos asyncLink seleccionados basada en el tipo de enlace. Para cambiar el valor de una de estas propiedades, seleccione el nombre de registro. En la lista de Propiedades de elementos asyncLink seleccionados, pulse una vez en la propiedad que desee definir para poner el campo de Valor asociado con esa propiedad en modalidad de edición. Defina el nuevo valor seleccionando una opción en la lista desplegable de Valor, o tecleando el nuevo valor en el campo Valor. Para algunas propiedades solamente puede seleccionar una opción en una lista desplegable. Para otras propiedades solamente puede teclear un valor en el campo Valor.
8. Modifique la lista de elementos asyncLink como sea necesario:
  - Para volver a situar un elemento asyncLink, seleccione un elemento y pulse **Colocar antes** o **Colocar después**.
  - Para eliminar un elemento asyncLink, seleccione un elemento y pulse en **Eliminar** o pulse la tecla Suprimir.

#### **Conceptos relacionados**

“Componente de opciones de enlace” en la página 311

#### **Tareas relacionadas**

“Añadir un componente de opciones de enlace a archivo construcción EGL” en la página 313

“Editar el elemento callLink de un componente de opciones de enlace” en la página 314

“Editar elementos relacionados con transfer. de comp. opciones enlace” en la página 317

“Eliminar componente opciones enlace a archivo de construcción de EGL” en la página 318

#### **Consulta relacionada**

“Elemento asyncLink” en la página 377

“Formato de un archivo de construcción EGL” en la página 380

“Archivo de propiedades de enlace (detalles)” en la página 657

“startTransaction()” en la página 911

## **Editar elementos relacionados con transfer. de comp. opciones enlace**

Un componente de opciones de enlace describe cómo un programa EGL generado implementa llamadas y transferencias y cómo el programa accede a archivos. Para editar los elementos relacionados con transferencias del componente, haga lo siguiente:

1. Para abrir un archivo de construcción EGL con el editor de componentes de construcción EGL, haga lo siguiente en el Explorador de proyectos:
  - a. Pulse con el botón derecho del ratón en el archivo de construcción de EGL
  - b. Seleccione **Abrir con > Editor de componentes de construcción de EGL**
2. Si la vista Esquema no se visualiza, ábrala seleccionando **Mostrar vista > Esquema** en el menú Ventana.
3. En la vista Esquema, pulse con el botón derecho del ratón en un componente de opciones de enlace y pulse en **Abrir**. El editor de componentes de construcción de EGL visualiza la declaración de componente actual.
4. Pulse en el botón Mostrar elementos TransferLink en la barra de herramientas del editor. Aparecerán las listas Transferir a programa y Transferir a transacción.
5. Para editar la lista Transferir a programa, haga lo siguiente:
  - a. Al final de la lista Transferir a programa, pulse en **Añadir** o pulse la tecla Insertar y teclee el nombre de Programa origen (fromPgm) o seleccione un nombre de programa en la lista desplegable Nombre de programa origen.
  - b. Para editar el nombre de Programa destino (toPgm), seleccione la fila correspondiente en la lista Transferir a programa y, a continuación, pulse una vez en el campo Programa destino para poner el campo en modalidad de edición. Teclee el nombre de programa o seleccione un nombre de programa en la lista desplegable Programa destino.
  - c. Si es necesario un nombre de alias, seleccione la fila correspondiente en la lista Transferir a programa y, a continuación, pulse una vez en el campo Alias para poner el campo en modalidad de edición. Teclee el nombre de alias.
  - d. Para cambiar el tipo de enlace por omisión asociado con el nombre de programa, seleccione la fila correspondiente en la lista Transferir a programa y, a continuación, pulse una vez en el campo Tipo de enlace (linkType) para poner el campo en modalidad de edición. Seleccione el nuevo tipo de enlace en la lista desplegable de Tipo de enlace.
6. Para editar la lista Transferir a transacción, haga lo siguiente:
  - a. Al final de la lista Transferir a transacción, pulse en **Añadir** o pulse la tecla Insertar y teclee el nombre de Programa destino (toPgm) o seleccione un nombre de programa en la lista desplegable Nombre de programa destino.
  - b. Si es necesario un nombre de alias, seleccione la fila correspondiente en la lista Transferir a transacción y, a continuación, pulse una vez en el campo Alias para poner el campo en modalidad de edición. Teclee el nombre de alias.
  - c. Para editar la propiedad Definido externamente asociada con el nombre de programa, seleccione la fila correspondiente en la lista Transferir a transacción y, a continuación, pulse una vez en el campo Definido

externamente para poner el campo en modalidad de edición. Seleccione la propiedad de definido externamente en la lista desplegable de la propiedad Definido externamente.

- d. Modifique la lista Transferir a transacción como sea necesario:
  - Para volver a situar un elemento transferToTransaction, seleccione un elemento y pulse **Colocar antes** o **Colocar después**.
  - Para eliminar un elemento transferToTransaction, seleccione un elemento y pulse en **Eliminar** o pulse la tecla Suprimir.

**Nota:** Transferir a programa solo es relevante en estos productos:

- Rational Application Developer para iSeries
- Rational Application Developer para z/OS

#### **Conceptos relacionados**

“Componente de opciones de enlace” en la página 311

#### **Tareas relacionadas**

“Añadir un componente de opciones de enlace a archivo construcción EGL” en la página 313

“Editar el elemento asynchLink de un componente de opciones de enlace” en la página 315

“Editar el elemento callLink de un componente de opciones de enlace” en la página 314

“Eliminar componente opciones enlace a archivo de construcción de EGL”

#### **Consulta relacionada**

“Formato de un archivo de construcción EGL” en la página 380

“Elemento transferToTransaction” en la página 952

### **Eliminar componente opciones enlace a archivo de construcción de EGL**

Para eliminar un componente de opciones de enlace de un archivo de construcción de EGL, haga lo siguiente:

1. Para abrir un archivo de construcción EGL con el editor de componentes de construcción EGL, haga lo siguiente en el Explorador de proyectos:
  - a. Pulse con el botón derecho del ratón en el archivo de construcción de EGL
  - b. Seleccione **Abrir con > Editor de componentes de construcción de EGL**
2. Si la vista Esquema no se visualiza, ábrala seleccionando **Mostrar vista > Esquema** en el menú Ventana
3. En la vista Esquema, pulse con el botón derecho del ratón sobre el componente de opciones de enlace y pulse **Eliminar**

#### **Conceptos relacionados**

“Componente de opciones de enlace” en la página 311

#### **Tareas relacionadas**

“Añadir un componente de opciones de enlace a archivo construcción EGL” en la página 313

“Editar el elemento asynchLink de un componente de opciones de enlace” en la página 315

“Editar el elemento callLink de un componente de opciones de enlace” en la página 314

“Editar elementos relacionados con transfer. de comp. opciones enlace” en la página 317

## Configurar referencias a otros archivos de construcción de EGL

### Añadir una sentencia de importación a un archivo de construcción EGL

Las sentencias de importación permiten a los archivos de construcción de EGL hacer referencia a otros componentes en otros archivos de construcción. Consulte *Importar* para obtener más información sobre la característica de importación.

Para añadir una sentencia de importación a un archivo de construcción EGL, haga lo siguiente:

1. Abra un archivo de construcción de EGL con el editor de componentes de construcción de EGL. Si no tiene un archivo abierto, haga lo siguiente en el Explorador de proyectos:
  - a. Pulse con el botón derecho del ratón en el Explorador de proyectos
  - b. Seleccione **Abrir con > Editor de componentes de construcción de EGL**
2. Pulse en la pestaña **Importaciones** en el editor de componentes de construcción.
3. Pulse en el botón **Añadir**.
4. Teclee o seleccione el nombre del archivo o carpeta a importar y, a continuación, pulse en **Aceptar**.

#### Conceptos relacionados

“Import” en la página 33

#### Tareas relacionadas

“Editar una sentencia de importación en un archivo de construcción EGL”

“Eliminar una sentencia de importación de un archivo de construcción EGL” en la página 320

### Editar una sentencia de importación en un archivo de construcción EGL

Para editar una sentencia de importación en un archivo de construcción EGL, haga lo siguiente:

1. Abra el archivo de construcción de EGL con el editor de componentes de construcción de EGL. Si no tiene un archivo abierto, haga lo siguiente en el Explorador de proyectos:
  - a. Pulse con el botón derecho del ratón en el Explorador de proyectos
  - b. Seleccione **Abrir con > Editor de componentes de construcción de EGL**
2. Pulse en la pestaña **Importaciones** en el editor de componentes de construcción. Se visualizan las sentencias de importación.
3. Seleccione la sentencia de importación que desea cambiar y pulse el botón **Editar**.
4. Teclee o seleccione el nombre del archivo o carpeta a importar y, a continuación, pulse en **Aceptar**.

#### Conceptos relacionados

“Import” en la página 33

#### Tareas relacionadas

“Añadir una sentencia de importación a un archivo de construcción EGL”

“Eliminar una sentencia de importación de un archivo de construcción EGL” en la página 320

## Eliminar una sentencia de importación de un archivo de construcción EGL

Para eliminar una sentencia de importación de un archivo de construcción EGL, haga lo siguiente:

1. Abra el archivo de construcción de EGL con el editor de componentes de construcción de EGL. Si no tiene un archivo abierto, haga lo siguiente en el Explorador de proyectos:
  - a. Pulse con el botón derecho del ratón en el Explorador de proyectos
  - b. Seleccione **Abrir con > Editor de componentes de construcción de EGL**
2. Pulse en la pestaña **Importaciones** en el editor de componentes de construcción. Se visualizan las sentencias de importación.
3. Seleccione la sentencia de importación que desea eliminar y pulse el botón **Eliminar**.

### Conceptos relacionados

“Import” en la página 33

### Tareas relacionadas

“Añadir una sentencia de importación a un archivo de construcción EGL” en la página 319

“Editar una sentencia de importación en un archivo de construcción EGL” en la página 319

---

## Editar un vía de acceso de construcción de EGL

Para obtener una visión general, consulte estos temas:

- *Referencias a componentes*
- *Vía de acceso de construcción de EGL y eglpath*

Para incluir proyectos en la vía de acceso de proyecto EGL, siga estos pasos:

1. En el Explorador de proyectos, pulse con el botón derecho del ratón en un proyecto que desee enlazar con otros proyectos y, a continuación, pulse en **Propiedades**.
2. Seleccione la página de propiedades de **Vía de acceso de construcción de EGL**.
3. En la pestaña **Proyectos** se visualizará una lista de todos los proyectos del espacio de trabajo. Pulse en el recuadro de selección junto a cada proyecto que desee referenciar.
4. Para colocar los proyectos en un orden distinto o para exportar alguno, pulse en la pestaña **Ordenar y exportar** y haga lo siguiente:
  - Para cambiar la posición de un proyecto en el orden de la vía de acceso de construcción, seleccione el proyecto y pulse en los botones **Arriba** y **Abajo**.
  - Para exportar un proyecto, seleccione el recuadro de selección relacionado. Para gestionar todos los proyectos a la vez, pulse en el botón **Seleccionar todo** o **Deseleccionar todo**.
5. Pulse en **Aceptar**.

### Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13

“Referencias a componentes” en la página 21

“Import” en la página 33

“Componentes” en la página 17

### Consulta relacionada

“Vía de acceso de construcción EGL y eglpath” en la página 477

"Referencias a componentes" en la página 21  
"Import" en la página 33  
"Componentes" en la página 17



---

# Generar, prepara y ejecutar salida de EGL

---

## Generación

La generación es la creación de salida a partir de componentes EGL.

Puede generar la salida en el entorno de trabajo, desde la interfaz por lotes del entorno de trabajo, o bien desde el Kit de desarrollo de software de EGL (SDK de EGL). El SDK de EGL proporciona una interfaz por lotes para la generación basada en archivo que es independiente del entorno de trabajo.

La generación utiliza las versiones guardadas de los archivos EGL.

### Conceptos relacionados

“Proceso de desarrollo” en la página 8

“Salida generada” en la página 529

### Tareas relacionadas

“Generar envolturas Java” en la página 300

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“Salida generada (referencia)” en la página 530

## Generación de código Java en un proyecto

Si está generando un programa o envoltura Java, es recomendable (y en algunos casos necesario) establecer la opción del descriptor de construcción **genProject**, que provoca la generación en un proyecto.

EGL proporciona diversos servicios al generar en un proyecto. Los servicios varían según el tipo de proyecto, al igual que las tareas siguientes:

### Proyecto de cliente de aplicaciones

Al generar en un proyecto de cliente de aplicaciones, EGL hace lo siguiente:

- Proporciona acceso en tiempo de preparación a archivos jar EGL (fda6.jar y fdaj6.jar) añadiendo las siguientes entradas a la vía de construcción Java del proyecto:

```
EGL_GENERATORS_PLUGINDIR/runtime/fda6.jar
EGL_GENERATORS_PLUGINDIR/runtime/fdaj6.jar
```

Para obtener detalles sobre la variable al principio de cada entrada, consulte *Establecer la variable EGL\_GENERATORS\_PLUGINDIR*.

- Proporciona acceso en tiempo de ejecución a los archivos jar EGL:
  - Importa los archivos jar a cada proyecto de aplicación de empresa que hace referencia al proyecto de cliente de aplicaciones
  - Actualiza el manifiesto en el proyecto de cliente de aplicaciones de forma que los archivos jar de un proyecto de aplicación de empresa estén disponibles
- Coloca valores de ejecución en el descriptor de despliegue para que pueda evitar tener que cortar y pegar entradas desde un archivo de entorno J2EE generado; para obtener una visión general de este tema, consulte *Establecer valores de descriptor de despliegue*

Las siguientes tareas son las indicadas a continuación:

1. Si está llamando al programa generado mediante TCP/IP, proporcione acceso de ejecución a un escucha, tal como se describe en *Configuración del escucha de TCP/IP*
2. Proporciona acceso a archivos jar no de EGL
3. Ahora que ha colocado archivos de salida en un proyecto, continúe configurando el entorno de tiempo de ejecución de J2EE

### Proyecto EJB

Al generar en un proyecto EJB, EGL hace lo siguiente:

- Proporciona acceso en tiempo de preparación a archivos jar EGL (fda6.jar y fdaj6.jar) añadiendo las siguientes entradas a la vía de construcción Java del proyecto:

```
EGL_GENERATORS_PLUGINDIR/runtime/fda6.jar
EGL_GENERATORS_PLUGINDIR/runtime/fdaj6.jar
```

Para obtener detalles sobre la variable de entorno al principio de cada entrada, consulte *Establecer la variable EGL\_GENERATORS\_PLUGINDIR*.

- Proporciona acceso en tiempo de ejecución a los archivos jar EGL:
  - Importa fda6.jar y fdaj6.jar a cada proyecto de aplicación de empresa que hace referencia al proyecto EJB
  - Actualiza el manifiesto en el proyecto EJB de forma que fda6.jar y fdaj6.jar de un proyecto de aplicación de empresa estén disponibles durante la ejecución
- Asigna el nombre JNDI automáticamente de forma que el código de ejecución de EGL pueda acceder al código EJB; este paso se produce solamente al generar un bean de sesión EJB.
- En la mayoría de casos, coloca valores de ejecución en el descriptor de despliegue de forma que pueda evitar tener que cortar y pegar entradas desde un archivo de entorno J2EE generado; para obtener una visión general de este tema, consulte *Establecer valores de descriptor de despliegue*.

EGL no coloca valores de ejecución en el descriptor de despliegue si EGL no encuentra el elemento de sesión necesario en el descriptor de despliegue. Esta situación se produce, por ejemplo, cuando el programa Java se genera antes que la envoltura o cuando la opción del descriptor de construcción **sessionBeanID** está establecida en un valor que no se encuentra en el descriptor de despliegue. Encontrará los detalles sobre los elementos de sesión en *sessionBeanID*.

Las siguientes tareas son las indicadas a continuación:

1. Proporcionar acceso a archivos jar no de EGL
2. Generar código de despliegue
3. Ahora que ha colocado archivos de salida en un proyecto, continúe configurando el entorno de tiempo de ejecución de J2EE

### Proyecto Web J2EE

EGL hace lo siguiente:

- Proporciona acceso a archivos jar EGL importando fda6.jar y fdaj6.jar a la carpeta Web Content/WEB-INF/lib del proyecto
- Coloca valores de ejecución en el descriptor de despliegue para que pueda evitar tener que cortar y pegar entradas desde un archivo de entorno J2EE generado; para obtener una visión general de este tema, consulte *Establecer valores de descriptor de despliegue*

Las siguientes tareas son las indicadas a continuación:

1. Proporcionar acceso a archivos jar no de EGL
2. Ahora que ha colocado archivos de salida en un proyecto, continúe tal como se ha descrito en la sección *Configurar el entorno de tiempo de ejecución J2EE para el código generado en EGL*

### Proyecto Java

Si está generando en un proyecto Java no de J2EE para depuración o producción, EGL hace lo siguiente:

- Proporciona acceso a archivos jar EGL (fda6.jar y fdaj6.jar) añadiendo las siguientes entradas a la vía de construcción Java del proyecto:

```
EGL_GENERATORS_PLUGINDIR/runtime/fda6.jar
EGL_GENERATORS_PLUGINDIR/runtime/fdaj6.jar
```

Para obtener detalles sobre la variable al principio de cada entrada, consulte *Establecer la variable EGL\_GENERATORS\_PLUGINDIR*.

- Genera un archivo de propiedades, pero solo si el descriptor de despliegue incluye los valores de opción siguientes:
  - **genProperties** está establecido en GLOBAL o PROGRAM; y
  - **J2EE** está establecido en NO.

Si solicita un archivo de propiedades global (**rununit.properties**), EGL coloca ese archivo en la carpeta fuente Java, que es la carpeta que contiene los paquetes Java. (La carpeta fuente Java puede ser una carpeta dentro del proyecto o el proyecto mismo.) Si en su lugar solicita un archivo de propiedades de programa, EGL coloca ese archivo con el programa.

Durante la ejecución, se utilizan los valores del archivo de propiedades de programa para establecer una conexión JDBC estándar. Encontrará los detalles en *Cómo se realiza una conexión JDBC estándar*.

Ahora que ha colocado archivos de salida en un proyecto, haga lo siguiente:

- Si su programa accede a una base de datos relacional, asegúrese de que la vía de construcción Java incluye el directorio en el que está instalado el controlador. Para DB2, por ejemplo, especifique el directorio que contiene db2java.zip.
- Si el código accede a MQSeries, proporciona acceso a archivos jar no de EGL
- Coloque un archivo de propiedades de enlace en el módulo

Para obtener detalles sobre las consecuencias de generar en un proyecto no existente, consulte *genProject*.

### Tareas relacionadas

“Generar código de despliegue para proyectos EJB” en la página 338

“Desplegar un archivo de propiedades de enlace” en la página 364

“Establecer valores de descriptor de despliegue” en la página 355

“Proporcionar acceso a archivos jar no de EGL” en la página 365

“Establecer la variable EGL\_GENERATORS\_PLUGINDIR” en la página 339

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

“Cómo se realiza una conexión JDBC estándar” en la página 263

### Consulta relacionada

“genProject” en la página 393

“sessionBeanID” en la página 398

## Construcción

Cuando se trabaja en un proyecto EGL o Web EGL, la palabra *construir* no se refiere (por lo general) a la generación de código.

Las siguientes opciones de menú tienen un significado diferenciado:

### Construir proyecto

Construye un subconjunto del proyecto:

1. Valida todos los archivos EGL que han cambiado en el proyecto desde la última construcción
2. Genera PageHandlers que se han cambiado desde la anterior generación de PageHandlers.
3. Compila el código fuente Java que ha cambiado desde la última compilación

La opción de menú **Construir proyecto** solamente está disponible si no se ha establecido la preferencia del entorno de trabajo **Realizar construcción automáticamente en la modificación de recursos**. Si *ha establecido* esa preferencia, las acciones descritas anteriormente se producen siempre que guarde un archivo EGL.

### Construir todos

Lleva a cabo las mismas acciones que **Construir proyecto**, pero para cada proyecto abierto en el espacio de trabajo.

### Volver a construir proyecto

Actúa de la manera siguiente:

1. Valida todos los archivos EGL del proyecto
2. Genera todos los PageHandlers del proyecto
3. Compila el código fuente Java que ha cambiado desde la última compilación

### Volver a construir todos

Lleva a cabo las mismas acciones que **Volver a construir proyecto**, pero para cada proyecto abierto en el espacio de trabajo.

Cuando se genera código en un proyecto, se realiza una compilación Java en las situaciones siguientes:

- Cuando se construye o se vuelve a construir el proyecto; o bien
- Cuando se generan los archivos fuente; pero sólo si se ha marcado la preferencia del entorno de trabajo **Realizar construcción automáticamente en modificación de recurso**.

Cuando se genera código en un directorio, EGL opcionalmente crea un *plan de construcción*, que es un archivo XML que incluye los siguientes detalles:

- La ubicación de los archivos que se transferirán a otra máquina;
- Otra información necesaria para la transferencia, que se realiza mediante TCP/IP; y
- Una sentencia de compilación Java.

La preparación de la salida generada en una plataforma remota requiere que un servidor de construcción se ejecute en dicha plataforma.

Es posible que desee crear un plan de construcción e invocar dicho plan más adelante. Para obtener información detallada, consulte la sección *Invocar un plan de construcción después de la generación*.

#### Conceptos relacionados

“Componente descriptor de construcción” en la página 293  
“Plan de construcción”

“Servidor de construcción” en la página 343  
“Proceso de desarrollo” en la página 8

#### Tareas relacionadas

“Crear un archivo de construcción” en la página 293  
“Invocar un plan de construcción tras la generación” en la página 334

#### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

## Construir la salida de EGL

Para construir la salida de EGL para programas o envolturas Java, complete los pasos siguientes:

1. Generar código fuente Java en un proyecto o directorio:
  - Si genera en un proyecto (el método recomendado) y las preferencias de Eclipse están establecidas para construir automáticamente en la modificación de recursos, el entorno de trabajo prepara la salida.
  - Si genera en un directorio, la función de construcción distribuida del generador prepara la salida.
2. Preparar la salida generada. Este paso se realiza automáticamente a menos que establezca la opción del descriptor de construcción **buildPlan** o **prep** en no.

#### Conceptos relacionados

“Construcción” en la página 326  
“Proceso de desarrollo” en la página 8  
“Proyectos, paquetes y archivos EGL” en la página 13  
“Generación de código Java en un proyecto” en la página 323  
“Salida generada” en la página 529  
“Generación” en la página 323

#### Tareas relacionadas

“Crear un archivo fuente EGL” en la página 128  
“Procesar código Java generado en un directorio” en la página 335  
“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

#### Consulta relacionada

“Salida generada (referencia)” en la página 530

## Plan de construcción

El plan de construcción es un archivo XML que proporciona los siguientes detalles durante la preparación:

- Qué archivos deben procesarse en la máquina de construcción
- Qué scripts de construcción se necesitan para procesarlos
- Dónde deben colocarse las salidas

El plan de construcción reside en la plataforma de desarrollo e informa al cliente de construcción de todos los pasos de la construcción. Para cada paso, se realiza una petición al servidor de construcción.

EGL genera un plan de construcción siempre que se genera un programa o envoltura Java, a menos que establezca la opción del descriptor de construcción **buildPlan** en NO.

Para obtener información detallada sobre el nombre del plan de construcción, consulte la sección *Salida generada (referencia)*.

#### **Conceptos relacionados**

“Script de construcción” en la página 342

“Salida generada” en la página 529

#### **Consulta relacionada**

“buildPlan” en la página 385

“Salida generada (referencia)” en la página 530

## **Programa Java, PageHandler y biblioteca**

Cuando se solicita que un componente de programa se genere como un programa Java o cuando se solicita que se genere un componente pageHandler o de biblioteca relacionado con Java, EGL genera una clase y un archivo para cada uno de los siguientes:

- El componente de programa, pageHandler o de biblioteca
- Cada registro declarado en el propio componente o en cualquier función invocada directa o indirectamente por dicho componente
- Cada tabla de datos, grupo de formularios y formulario que se utiliza

Para obtener información detallada sobre los nombres de clase, consulte la sección *Salida de la generación de programas Java*.

#### **Tareas relacionadas**

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

#### **Consulta relacionada**

“Salida generada (referencia)” en la página 530

“Salida de la generación de programa Java” en la página 675

## **Archivo de resultados**

El archivo de resultados contiene información de estado sobre los pasos de preparación de código que se han realizado en el entorno destino. Sólo se recibe el archivo si EGL intenta preparar la salida generada.

La preparación se produce automáticamente cuando se genera en un directorio y se utilizan las siguientes opciones del descriptor de construcción:

- **prep** está establecida en YES
- **buildPlan** está establecida en YES

Para obtener información detallada sobre el nombre del archivo de resultados, consulte la sección *Salida generada (referencia)*.

### Conceptos relacionados

“Componente descriptor de construcción” en la página 293

“Salida generada” en la página 529

### Tareas relacionadas

“Procesar código Java generado en un directorio” en la página 335

### Consulta relacionada

“Salida generada (referencia)” en la página 530

“buildPlan” en la página 385

“prep” en la página 397

---

## Generar en el entorno de trabajo

Generar en el entorno de trabajo se consigue mediante el asistente para la generación o el elemento de menú generación. Al seleccionar el elemento de menú generación, EGL utiliza el descriptor de construcción por omisión. Si no ha seleccionado un descriptor de construcción por omisión, utilice el asistente para la generación. Para conocer detalles sobre la selección del descriptor de construcción por omisión, consulte *Establecer los descriptores de construcción por omisión*.

Para generar en el entorno de trabajo invocando al asistente para la generación, haga lo siguiente:

1. Pulse con el botón derecho del ratón en un nombre de recurso (proyecto, carpeta o archivo) en el Explorador de proyectos.
2. Seleccione la opción **Generar con asistente...**

El asistente para la generación incluye cuatro páginas:

1. La primera página visualiza una lista de componentes a generar basada en la selección que realizó el usuario para iniciar el proceso de generación. Debe seleccionar al menos un componente de la lista para poder continuar en la página siguiente. La interfaz proporciona botones para permitirle seleccionar o deseleccionar todos los componentes de la lista.
2. La segunda página le permite elegir un descriptor o descriptores de construcción a utilizar para generar los componentes seleccionados en la primera página. Tiene dos opciones:
  - Elija de una lista desplegable de todos los descriptores de construcción que hay en el espacio de trabajo y utilice ese descriptor de construcción para generar todos los componentes.
  - Seleccione un descriptor de construcción para cada uno de los componentes seleccionados en la primera página. Se utiliza una tabla para seleccionar los descriptores de construcción para cada componente. La primera columna de la tabla visualiza los nombres de componentes; la segunda columna visualiza una lista desplegable de descriptores de construcción para cada componente.
3. La tercera página le permite establecer los ID de usuario y contraseñas para la máquina destino y la base de datos de SQL utilizadas en el proceso de generación, si son necesarios ID de usuario y contraseñas. Estos ID de usuario y contraseñas, si los hay, alteran temporalmente los listados en el descriptor de construcción especificado para cada componente que se genera. Puede interesarle establecer los ID de usuario y contraseñas en esta página en lugar de en el descriptor de construcción para evitar conservar la información confidencial en almacenamiento persistente.
4. La cuarta página le permite crear un archivo de mandatos que puede utilizar para generar un programa EGL fuera del entorno de trabajo. Puede hacer

referencia al archivo de mandatos en la interfaz por lotes del entorno de trabajo (utilizando el mandato EGLCMD) o en el SDK de EGL (utilizando el mandato EGLSDK).

Para crear un archivo de mandatos, haga lo siguiente:

- a. Seleccione el recuadro de selección Crear un archivo de mandatos
- b. Especifique el nombre del archivo de salida, ya sea tecleando la vía de acceso totalmente calificada o pulsando en **Examinar** y utilizando el procedimiento estándar de Windows para seleccionar un archivo
- c. Seleccione o deseleccione el recuadro de selección Insertar vía de acceso de EGL automáticamente (eglpath) para especificar si desea incluir la vía de acceso del proyecto de EGL en el archivo de mandatos, como valor inicial para eglpath; encontrará los detalles en *Archivo de mandatos EGL*
- d. Seleccione un botón de selección para indicar si debe evitarse generar salida al crear el archivo de mandatos

5. Pulse en **Finalizar**.

Para generar en el entorno de trabajo utilizando el elemento de menú generación, siga uno de las siguientes series de pasos:

1. Seleccione uno o más nombres de recurso (proyecto, carpeta o archivo) en el Explorador de proyectos. Para seleccionar múltiples nombres de recurso, mantenga pulsada la tecla **Control** mientras pulsa en ellos.
2. Pulse con el botón derecho del ratón y seleccione la opción de menú **Generar**.

o bien

1. Pulse dos veces en un nombre de recurso (proyecto, carpeta o archivo) en el Explorador de proyectos. El archivo se abrirá en el editor EGL.
2. Pulse con el botón derecho del ratón dentro del panel del editor y seleccione **Generar**.

#### **Conceptos relacionados**

“Generación en el entorno de trabajo”

#### **Tareas relacionadas**

“Establecer los descriptores de construcción por omisión” en la página 116

#### **Consulta relacionada**

“EGLCMD” en la página 478

“EGLSDK” en la página 488

“Salida generada (referencia)” en la página 530

## **Generación en el entorno de trabajo**

Para generar la salida en el entorno de trabajo, realice lo siguiente:

- Cargue los componentes que deben generarse, junto con los componentes a los que se hace referencia durante la generación.
- Seleccione los componentes que deben generarse. Si invoca el proceso de generación para un archivo, carpeta, paquete o proyecto, EGL crea salida para cada *componente primario* (cada programa, PageHandler, grupo de formularios, tabla de datos, o biblioteca) que está en el contenedor que ha seleccionado.
- Inicie la generación.
- Supervise el progreso.

Para facilitar la generación, se recomienda que primero seleccione un descriptor de construcción por omisión de los siguientes tipos:

- Descriptor de construcción de depuración (según convenga cuando utilice el depurador EGL)
- Descriptor de construcción de sistema destino (tal como se utiliza para generar componentes y desplegarlos en un entorno de ejecución)

Para obtener información detallada sobre cómo seleccionar un descriptor de construcción por omisión, consulte la sección *Establecer los descriptores de construcción por omisión*.

Puede identificar los dos descriptores de construcción (depuración y sistema destino) de las siguientes maneras:

- Como una propiedad a nivel de archivo, carpeta, paquete y proyecto
- Como una preferencia del entorno de trabajo

Un descriptor de construcción de nivel inferior de un determinado tipo tiene preferencia sobre cualquier descriptor de construcción de nivel superior del mismo tipo. Por ejemplo, un descriptor de construcción de sistema destino que se ha asignado al paquete actual tiene preferencia sobre un descriptor de construcción de sistema destino que se ha asignado al proyecto. Sin embargo, un descriptor de construcción maestro tiene preferencia sobre todos los demás, como se describe en la sección *Componente descriptor de construcción*.

Se aplican las siguientes normas de preferencia para cada archivo que contiene un componente primario:

- Una propiedad que es específica del archivo tiene preferencia sobre todas las demás
- La propiedad relacionada de carpeta tiene preferencia sobre la propiedad de paquete, proyecto o entorno de trabajo
- La propiedad relacionada de paquete tiene preferencia sobre la propiedad de proyecto o entorno de trabajo
- La propiedad relacionada de proyecto tiene preferencia sobre la propiedad de entorno de trabajo
- La propiedad de entorno de trabajo se utiliza si no se especifican otras

Para obtener información detallada sobre cómo iniciar la generación, consulte la sección *Generar en el entorno de trabajo*.

#### **Conceptos relacionados**

“Componente descriptor de construcción” en la página 293

“Proceso de desarrollo” en la página 8

“Salida generada” en la página 529

#### **Tareas relacionadas**

“Generar en el entorno de trabajo” en la página 329

“Establecer los descriptores de construcción por omisión” en la página 116

#### **Consulta relacionada**

“Salida generada (referencia)” en la página 530

---

## **Generar desde la interfaz por lotes del entorno de trabajo**

Para generar desde la interfaz por lotes del entorno de trabajo, haga lo siguiente:

1. Asegúrese de que la vía de acceso de clases de Java proporciona acceso a estos archivos jar:
  - startup.jar, que está en el directorio siguiente:  
`dirInstalación\eclipse`  
*dirInstalación*  
 El directorio de instalación del producto, como por ejemplo C:\Program Files\IBM\RSPD\6.0. Si instaló y tuvo un producto de Rational Developer antes de instalar el producto que está utilizando ahora, deberá especificar el directorio utilizado en la instalación anterior.
  - eglutil.jar, que está en el directorio siguiente:  
`dirInstalación\egl\eclipse\plugins\com.ibm.etools.egl.utilities_versión\runtime`  
*dirInstalación*  
 El directorio de instalación del producto, como por ejemplo C:\Program Files\IBM\RSPD\6.0. Si instaló y tuvo un producto de Rational Developer antes de instalar el producto que está utilizando ahora, deberá especificar el directorio utilizado en la instalación anterior.
  - versión*  
 La versión instalada del conector; por ejemplo, 6.0.0
2. Asegúrese de que un espacio de trabajo contiene los proyectos y componentes de EGL que son necesarios para la generación.
3. Desarrolle un archivo de mandatos de EGL.
4. Invoque el mandato EGLCMD, si es posible en un trabajo por lotes más grande que genere, ejecute y pruebe el código. Al invocar a EGLCMD, especificará el espacio de trabajo que le interesa.

#### **Conceptos relacionados**

“Generación a partir de la interfaz por lotes del entorno de trabajo”

#### **Consulta relacionada**

“EGLCMD” en la página 478

“Archivo de mandatos EGL” en la página 481

## **Generación a partir de la interfaz por lotes del entorno de trabajo**

La interfaz por lotes del entorno de trabajo es una característica que permite generar salida EGL a partir de un entorno por lotes que puede acceder al entorno de trabajo. No es necesario que el entorno de trabajo esté en ejecución. La generación de código EGL sólo puede acceder a proyectos y componentes EGL que se han cargado previamente en un espacio de trabajo.

Para invocar la interfaz, utilice el mandato por lotes EGLCMD, que hace referencia a un espacio de trabajo y a un archivo de mandatos EGL.

#### **Conceptos relacionados**

“Proceso de desarrollo” en la página 8

“Salida generada” en la página 529

#### **Tareas relacionadas**

“Generar desde la interfaz por lotes del entorno de trabajo” en la página 331

### Consulta relacionada

“EGLCMD” en la página 478

---

## Generar a partir del SDK (Software Development Kit) de EGL

Para generar desde el SDK de EGL, haga lo siguiente:

1. Asegúrese de que Java 1.3.1 (o un nivel superior) está en la máquina en la que va a generar código. Se instala un nivel adecuado de código Java automáticamente en la máquina en la que instala EGL. Los niveles de Java en las máquinas de generación y de destino deben ser compatibles.

2. Asegúrese de que eglbatchgen.jar está en la vía de acceso de clases de Java. El archivo jar está en el directorio siguiente:

*dirInstalación*\bin

*dirInstalación*

El directorio de instalación del producto, como por ejemplo C:\Program Files\IBM\RSPD\6.0. Si instaló y tuvo un producto de Rational Developer antes de instalar el producto que está utilizando ahora, deberá especificar el directorio utilizado en la instalación anterior.

3. Asegúrese de que el SDK de EGL puede acceder a los archivos de EGL que son necesarios para la generación

4. Opcionalmente, desarrolle un archivo de mandatos EGL

5. Invoque el mandato EGLSDK, si es posible en un trabajo por lotes más grande que genere, ejecute y pruebe el código

### Conceptos relacionados

“Generación a partir del SDK (Software Development Kit) de EGL”

“Proyectos, paquetes y archivos EGL” en la página 13

### Consulta relacionada

“Vía de acceso de construcción EGL y eglpath” en la página 477

“EGLCMD” en la página 478

“Archivo de mandatos EGL” en la página 481

“EGLSDK” en la página 488

## Generación a partir del SDK (Software Development Kit) de EGL

El Kit de desarrollo de software (SDK) de EGL es una característica que le permite generar la salida en un entorno por lotes, incluso cuando no se tiene acceso a los siguientes aspectos del producto Rational Developer:

- La interfaz gráfica de usuario
- Los detalles sobre cómo están organizados los proyectos

Puede utilizar el SDK de EGL para desencadenar la generación a partir de una herramienta de gestión de configuraciones de software (SCM), como por ejemplo Rational ClearCase, quizás como parte de un trabajo por lotes que se ejecuta después del horario trabajo normal.

Para invocar el SDK de EGL, utilice el mandato EGLSDK en un archivo por lotes o en un indicador de mandatos. La propia invocación de mandatos puede adoptar una de las dos formas siguientes:

- Puede especificar un descriptor de construcción y un archivo EGL. En este caso, si desea que se produzcan varias generaciones debe escribir varios mandatos.

- O bien, la invocación puede hacer referencia a un archivo de mandatos EGL que incluye la información necesaria para producir una o más generaciones.

Independientemente de cómo organice el trabajo, puede especificar un valor para *eglp**path*, que es una lista de directorios donde se realiza una búsqueda cuando el SDK de EGL utiliza una sentencia import para resolver una referencia a un componente. Además, debe especificar la opción del descriptor de construcción **genDirectory** en lugar de **genProject**.

Los requisitos previos y el proceso para utilizar EGLSDK se describen en la sección *Generar a partir del SDK de EGL*. Para obtener información detallada sobre la invocación de mandatos, consulte la sección *EGLSDK*.

#### Conceptos relacionados

“Proceso de desarrollo” en la página 8

“Salida generada” en la página 529

#### Tareas relacionadas

“Generar a partir del SDK (Software Development Kit) de EGL” en la página 333

#### Consulta relacionada

“genDirectory” en la página 391

“EGLCMD” en la página 478

“Vía de acceso de construcción EGL y *eglp**path*” en la página 477

“EGLSDK” en la página 488

---

## Invocar un plan de construcción tras la generación

Puede interesarle crear un plan de construcción e invocar ese plan más adelante. Este caso podría producirse, por ejemplo, si una anomalía de la red le impide preparar código en una máquina remota en el momento de la generación.

Para invocar un plan de construcción en este caso, complete los pasos siguientes:

1. Asegúrese de que *eglp**batchgen.jar* está en la vía de acceso de clases de Java, como sucede automáticamente en la máquina en la que instala EGL. El archivo *jar* está en el directorio siguiente:

```
dirInstalación\egl\eclipse\plugins\
com.ibm.etools.egl.batchgeneration_versión
```

*dirInstalación*

El directorio de instalación del producto, como por ejemplo C:\Program Files\IBM\RSPD\6.0. Si instaló y tuvo un producto de Rational Developer antes de instalar el producto que está utilizando ahora, deberá especificar el directorio utilizado en la instalación anterior.

*versión*

La versión instalada del conector; por ejemplo, 6.0.0

2. Igualmente, asegúrese de que la variable PATH incluye ese directorio.
3. Desde una línea de mandatos, entre el siguiente mandato:

```
java com.ibm.etools.egl.distributedbuild.BuildPlanLauncher bp
```

*bp* La vía de acceso totalmente calificado del archivo de plan de construcción. Para conocer los detalles sobre el nombre del archivo generado, consulte *Salida generada* (referencia).

### Conceptos relacionados

“Plan de construcción” en la página 327

“Generación” en la página 323

### Tareas relacionadas

“Construir la salida de EGL” en la página 327

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“Salida generada (referencia)” en la página 530

---

## Generar Java; varios temas

### Procesar código Java generado en un directorio

Esta página describe cómo procesar código que se genera en un directorio. Se recomienda, no obstante, que evite generar código en un directorio; para obtener detalles consulte la sección *Generación de código Java en un proyecto*.

Para generar código en un directorio, especifique la opción del descriptor de construcción **genDirectory** y evite especificar la opción del descriptor de construcción **genProject**.

Las siguientes tareas dependen del tipo de proyecto:

#### Proyecto de cliente de aplicaciones

Para un proyecto de cliente de aplicaciones, haga lo siguiente:

1. Proporcione acceso en tiempo de preparación a archivos jar EGL añadiendo las siguientes entradas a la vía de construcción Java del proyecto:

```
EGL_GENERATORS_PLUGINDIR/runtime/fda6.jar
EGL_GENERATORS_PLUGINDIR/runtime/fdaj6.jar
```

Para obtener detalles sobre la variable al principio de cada entrada, consulte *Establecer la variable EGL\_GENERATORS\_PLUGINDIR*.

2. Proporcione acceso en tiempo de ejecución a fda6.jar, fdaj6.jar y (si está llamado al programa generado mediante TCP/IP) EGLTcpiListener.jar:

- Acceda a los archivos jar desde el siguiente directorio:

```
dirInstalación\egl\eclipse\plugins\
com.ibm.etools.egl.generators_versión\runtime
```

#### *dirInstalación*

El directorio de instalación del producto, como por ejemplo C:\Program Files\IBM\RSPD\6.0. Si instaló y tuvo un producto de Rational Developer antes de instalar el producto que está utilizando ahora, deberá especificar el directorio utilizado en la instalación anterior.

#### *versión*

La versión instalada del conector; por ejemplo, 6.0.0

Copie esos archivos a cada proyecto de aplicación de empresa que haga referencia al proyecto de cliente de aplicaciones.

- Actualice el manifiesto en el proyecto de cliente de aplicaciones de forma que los archivos jar (almacenados en un proyecto de aplicación de empresa) estén disponibles.
3. Proporcione acceso a archivos jar no de EGL (una tarea opcional)
4. Importe la salida generada al proyecto, de acuerdo con estas normas:

- La carpeta *appClientModule* debe incluir la carpeta de nivel superior del paquete que contiene la salida generada
- La jerarquía de nombres de carpeta bajo *appClientModule* debe coincidir con el nombre de su paquete Java

Si está importando salida generada desde el paquete *my.trial.package*, por ejemplo, debe importar esa salida a una carpeta que resida en la siguiente ubicación:

```
appClientModule/my/trial/package
```

5. Si ha generado un archivo de entorno J2EE, actualice ese archivo
6. Actualice el descriptor de despliegue
7. Ahora que ha colocado archivos de salida en un proyecto, continúe configurando el entorno de tiempo de ejecución de J2EE

### Proyecto EJB

Para un proyecto EJB, haga lo siguiente:

1. Proporcione acceso en tiempo de preparación a archivos jar EGL (fda6.jar y fdaj6.jar) añadiendo las siguientes entradas a la vía de construcción Java del proyecto:

```
EGL_GENERATORS_PLUGINDIR/runtime/fda6.jar
EGL_GENERATORS_PLUGINDIR/runtime/fdaj6.jar
```

Para obtener detalles sobre la variable al principio de cada entrada, consulte *Establecer la variable EGL\_GENERATORS\_PLUGINDIR*.

2. Proporcione acceso en tiempo de ejecución a los archivos jar EGL:

- Acceda a fda6.jar y fdaj6.jar desde el directorio siguiente:

```
dirInstalación\egl\eclipse\plugins\
com.ibm.etools.egl.generators_versión\runtime
```

#### *dirInstalación*

El directorio de instalación del producto, como por ejemplo C:\Program Files\IBM\RSPD\6.0. Si instaló y tuvo un producto de Rational Developer antes de instalar el producto que está utilizando ahora, deberá especificar el directorio utilizado en la instalación anterior.

#### *versión*

La versión instalada del conector; por ejemplo, 6.0.0

Copie esos archivos a cada proyecto de aplicación de empresa que haga referencia al proyecto EJB.

- Actualice el manifiesto en el proyecto EJB de forma que fda6.jar y fdaj6.jar (almacenados en un proyecto de aplicación de empresa) estén disponibles.
3. Proporcione acceso a archivos jar no de EGL (una tarea opcional)
  4. Importe la salida generada al proyecto, de acuerdo con estas normas:
    - La carpeta *ejbModule* debe incluir la carpeta de nivel superior del paquete que contiene la salida generada
    - La jerarquía de nombres de carpeta bajo *ejbModule* debe coincidir con el nombre de su paquete Java

Si está importando salida generada desde el paquete *my.trial.package*, por ejemplo, debe importar esa salida a una carpeta que resida en la siguiente ubicación:

```
ejbModule/my/trial/package
```

5. Si ha generado un archivo de entorno J2EE, actualice ese archivo.

6. Actualice el descriptor de despliegue
7. Establezca el nombre JNDI
8. Genere código de despliegue
9. Ahora que ha colocado archivos de salida en un proyecto, continúe configurando el entorno de tiempo de ejecución de J2EE

### Proyecto Web J2EE

Para un proyecto Web, haga lo siguiente:

1. Proporcione acceso a archivos jar EGL copiando fda6.jar y fdaj6.jar en la carpeta del proyecto Web. Para hacerlo, importe los jar externos encontrados en el siguiente directorio:

```
dirInstalación\egl\eclipse\plugins\
com.ibm.etools.egl.generators_versión\runtime
```

#### *dirInstalación*

El directorio de instalación del producto, como por ejemplo C:\Program Files\IBM\RSPD\6.0. Si instaló y tuvo un producto de Rational Developer antes de instalar el producto que está utilizando ahora, deberá especificar el directorio utilizado en la instalación anterior.

#### *versión*

La versión instalada del conector; por ejemplo, 6.0.0

El destino de los archivos es la siguiente carpeta de proyecto:

```
WebContent/WEB-INF/lib
```

2. Proporcione acceso a archivos jar no de EGL (una opción)
3. Importe la salida generada al proyecto, de acuerdo con estas normas:
  - La carpeta *WebContent* debe incluir la carpeta de nivel superior del paquete que contiene la salida generada
  - La jerarquía de nombres de carpeta bajo *WebContent* debe coincidir con el nombre de su paquete Java

Si está importando salida generada desde el paquete *my.trial.package*, por ejemplo, debe importar esa salida a una carpeta que resida en la siguiente ubicación:

```
WebContent/my/trial/package
```

4. Actualice el descriptor de despliegue
5. Ahora que ha colocado archivos de salida en un proyecto, continúe configurando el entorno de tiempo de ejecución de J2EE

### Proyecto Java

Si está generando código para utilizarlo en un entorno no J2EE, puede generar un archivo de propiedades si utiliza la combinación de opciones de descriptor de construcción:

- **genProperties** está establecido en GLOBAL o PROGRAM; y
- **J2EE** está establecido en NO.

Si solicita un archivo de propiedades global (**rununit.properties**), EGL coloca ese archivo en el directorio de nivel superior. Si en su lugar solicita un archivo de propiedades de programa, EGL coloca el archivo con el programa, ya sea en la carpeta que corresponde al último calificador del nombre de paquete o en el directorio de nivel superior. (El directorio de nivel superior se utiliza si el nombre de paquete no se especifica en el archivo fuente EGL.)

Durante la ejecución, se utilizan los valores del archivo de propiedades de programa para establecer una conexión JDBC estándar. Encontrará los detalles en *Cómo se realiza una conexión JDBC estándar*.

Para un proyecto Java, las tareas son las siguientes:

1. Proporcione acceso a archivos jar EGL añadiendo las siguientes entradas a la vía de construcción Java del proyecto:  
EGL\_GENERATORS\_PLUGINDIR/runtime/fda6.jar  
EGL\_GENERATORS\_PLUGINDIR/runtime/fdaj6.jar  
Para obtener detalles sobre la variable al principio de cada entrada, consulte *Establecer la variable EGL\_GENERATORS\_PLUGINDIR*.
2. Si su programa accede a una base de datos relacional, asegúrese de que la vía de construcción Java incluye el directorio en el que está instalado el controlador. Para DB2, por ejemplo, especifique el directorio que contiene db2java.zip.
3. Si el código generado accede a MQSeries, proporcione acceso a archivos jar no de EGL
4. Asegúrese de que el archivo de propiedades del programa (si está presente) esté en la carpeta del proyecto de nivel superior y que el archivo de propiedades global (**rununit.properties**, si está presente) esté en la carpeta que corresponde al último calificador del nombre de paquete o en la carpeta del proyecto de nivel superior. (La carpeta de nivel superior se utiliza si el nombre de paquete no se especifica en el archivo fuente EGL.)
5. Coloque un archivo de propiedades de enlace en el proyecto (una tarea opcional)

#### Conceptos relacionados

“Generación de código Java en un proyecto” en la página 323

#### Tareas relacionadas

“Generar código de despliegue para proyectos EJB”

“Desplegar un archivo de propiedades de enlace” en la página 364

“Establecer valores de descriptor de despliegue” en la página 355

“Proporcionar acceso a archivos jar no de EGL” en la página 365

“Establecer el nombre JNDI para proyectos EJB” en la página 358

“Establecer la variable EGL\_GENERATORS\_PLUGINDIR” en la página 339

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

“Cómo se realiza una conexión JDBC estándar” en la página 263

“Actualizar el descriptor de despliegue manualmente” en la página 357

“Actualizar el archivo de entorno J2EE” en la página 356

#### Consulta relacionada

“genDirectory” en la página 391

“genProject” en la página 393

## Generar código de despliegue para proyectos EJB

Tras generar en un proyecto EJB y especificar las propiedades del descriptor de despliegue, puede generar los apéndices y esqueletos que permiten el acceso remoto de EJB:

1. En el Explorador de proyectos, pulse con el botón derecho del ratón sobre el nombre del proyecto y, a continuación, pulse **Desplegar**
2. Siga las instrucciones especificadas en la página de ayuda sobre Generar código de despliegue EJB desde el entorno de trabajo

#### Tareas relacionadas

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

## Establecer la variable `EGL_GENERATORS_PLUGINDIR`

La variable de vía de acceso de clases de entorno de trabajo `EGL_GENERATORS_PLUGINDIR` contiene la vía de acceso totalmente calificado al conector EGL en el Entorno de trabajo. La variable se utiliza en la vía de construcción Java al generar un programa EGL en un proyecto del tipo Java, cliente de aplicaciones o EJB.

Si encuentra un error de vía de acceso de clases que hace referencia a `EGL_GENERATORS_PLUGINDIR`, es posible que no se haya establecido la variable. El problema se produce, por ejemplo, si extrae un proyecto relacionado con EGL desde un sistema de gestión de configuraciones de software como el sistema de versiones concurrentes (CVS) antes de trabajar con un componente de EGL.

Puede establecer la variable creando un componente de EGL, generando código EGL o siguiendo estos pasos:

1. Seleccione **Ventana** y, a continuación, **Preferencias**
2. En la página Preferencias, seleccione **Java** y, a continuación, **Variables de vía de acceso de clases**
3. Seleccione **Nuevo...**
4. En la página Entrada de variable nueva, teclee `EGL_GENERATORS_PLUGINDIR` y especifique el siguiente directorio:

```
dirInstalación\egl\eclipse\plugins\
com.ibm.etools.egl.generators_versión
```

*dirInstalación*

El directorio de instalación del producto, como por ejemplo C:\Program Files\IBM\RSPD\6.0. Si instaló y tuvo un producto de Rational Developer antes de instalar el producto que está utilizando ahora, deberá especificar el directorio utilizado en la instalación anterior.

*versión*

La versión instalada del conector; por ejemplo, 6.0.0

Tras establecer la variable, vuelva a crear el proyecto.

#### Conceptos relacionados

“Generación de código Java en un proyecto” en la página 323

#### Consulta relacionada

“genProject” en la página 393

---

## Ejecutar código Java generado por EGL en la máquina local

### Iniciar una aplicación Java básica o de interfaz de usuario de texto en el sistema local

Para iniciar una aplicación Java básica (por lotes) o de interfaz de usuario de texto (TUI) generada por EGL en el sistema local, haga lo siguiente:

1. Genere código Java a partir del código fuente EGL; para conocer los detalles consulte la sección *Generar en el entorno de trabajo*.

2. En el Explorador de proyectos, expanda la carpeta **JavaSource** y seleccione el archivo fuente Java de la aplicación que desea ejecutar.
3. En el menú del entorno de trabajo, seleccione **Ejecutar > Ejecutar como > Aplicación Java**, o bien, en la barra de herramientas del entorno de trabajo, pulse la flecha abajo situada junto al botón **Ejecutar** y seleccione **Ejecutar como > Aplicación Java**.

#### Conceptos relacionados

“Generación en el entorno de trabajo” en la página 330

#### Tareas relacionadas

“Generar en el entorno de trabajo” en la página 329

## Iniciar una aplicación Web en el sistema local

Si está trabajando con una aplicación Web basada en EGL que accede a un origen de datos JNDI, no puede seguir las instrucciones del tema actual a menos que anteriormente haya configurado un servidor de aplicaciones Web. Para obtener información específica de WebSphere, consulte *WebSphere Application Server y EGL*.

Para iniciar una aplicación Web, siga estos pasos:

1. Genere código Java a partir del código fuente EGL; para conocer los detalles consulte la sección *Generar en el entorno de trabajo*.
2. En el Explorador de proyectos, expanda las carpetas **WebContent** y **WEB-INF**. Pulse el botón derecho del ratón sobre el JSP que desea ejecutar y seleccione **Ejecutar > Ejecutar en servidor**. Se visualiza el diálogo Selección de servidor.
3. Si ya ha configurado un servidor para este proyecto Web, seleccione **Elegir un servidor existente** y seleccione un servidor de la lista. Pulse **Finalizar** para iniciar el servidor, para desplegar la aplicación en el servidor y para iniciar la aplicación.
4. Si no ha configurado un servidor para este proyecto Web, puede continuar de la manera siguiente, pero solo si la aplicación no accede a un origen de datos JNDI:
  - a. Seleccione **Definir manualmente un servidor**.
  - b. Especifique el nombre de sistema principal que (para el sistema local) es **localhost**.
  - c. Seleccione un tipo de servidor que sea parecido al servidor de aplicaciones Web en el que pretende desplegar la aplicación en tiempo de ejecución. Las opciones incluyen **Entorno de prueba de WebSphere v5.1** y **WebSphere v6.0 Server**.
  - d. Si no pretende cambiar las opciones al trabajar en el proyecto actual, marque el recuadro de selección para **Establecer servidor como proyecto predeterminado**.
  - e. En la mayoría de los casos puede ahorrarse este paso, pero si desea especificar valores distintos de los predeterminados, pulse **Siguiente** y haga sus selecciones.
  - f. Pulse **Finalizar** para iniciar el servidor, para desplegar la aplicación en el servidor y para iniciar la aplicación.

#### Conceptos relacionados

“Generación en el entorno de trabajo” en la página 330

“WebSphere Application Server y EGL” en la página 341

“Soporte Web” en la página 187

## Tareas relacionadas

“Generar en el entorno de trabajo” en la página 329

## WebSphere Application Server y EGL

Cuando ejecute o depure una aplicación J2EE escrita en EGL en el entorno de trabajo, utilizará uno de estos entornos de tiempo de ejecución de IBM:

- Entorno de prueba de WebSphere v5.1 que soporta el servlet Java Versión 2.3 (y anteriores) y EJB Versión 2.0 (y anteriores)
- WebSphere Application Server v6.0, que soporta el servlet Java Versión 2.4 (y anteriores) y EJB Versión 2.1 (y anteriores)

Si está depurando o ejecutando código que no utiliza un origen de datos J2EE, los procesos para ejecutar código en los dos entornos son parecidos y solo requieren unas cuantas pulsaciones de ratón.

Sin embargo, si necesita acceder a un origen de datos J2EE, la situación es la siguiente:

- Si está trabajando con el Entorno de prueba de WebSphere v5.1, lleve a cabo los pasos siguientes en cualquier orden:
  1. Identifique el origen de datos cuando defina la configuración de servidor.
  2. Asegúrese de que la aplicación hace referencia a la entrada de configuración del servidor para ese origen de datos.

Este segundo paso incluye especificar el nombre JNDI en el descriptor de despliegue específico del proyecto. Puede especificar el nombre JNDI de cualquiera de estas maneras:

- Cuando crea el proyecto
- Al actualizar el descriptor de despliegue.

Para obtener detalles sobre la configuración del servidor, consulte la sección *Configurar WebSphere Application Server v5.x*.

- Si está trabajando con WebSphere Application Server v6.0, lleve a cabo los pasos siguientes en cualquier orden:
  1. Identifique el origen de datos con el servidor, de cualquiera de las maneras siguientes:
    - Cuando actualice el descriptor de despliegue de aplicaciones (application.xml), tal como se recomienda
    - Cuando configure el servidor en la Consola administrativa.

Para conocer detalles sobre la actualización del descriptor de despliegue de aplicaciones, consulte la sección *Configurar un servidor para probar orígenes de datos para WebSphere Application Server v6.0*. Para conocer detalles sobre cómo utilizar la Consola administrativa, consulte la sección *Configurar WebSphere Application Server v6.x*.

2. Asegúrese de que la aplicación hace referencia a la entrada de configuración del servidor para ese origen de datos.

Este segundo paso incluye especificar el nombre JNDI en el descriptor de despliegue específico del proyecto. Puede especificar el nombre JNDI de cualquiera de estas maneras:

- Cuando crea el proyecto
- Al actualizar el descriptor de despliegue.

Las ventajas de actualizar el descriptor de despliegue de aplicaciones en lugar de trabajar con la Consola administrativa son las siguientes:

- Puede desplegar la aplicación de empresa en cualquier aplicación de servidor Web que soporte J2EE Versión 1.4, sin la configuración de servidor adicional necesaria para identificar el origen de datos.
- Puede actualizar el descriptor de despliegue de aplicaciones independientemente de si el servidor está ejecutándose.
- Resulta ventajoso porque las acciones caen dentro del componente de desarrollo Rational Developer en lugar de dentro del componente WebSphere Application Server.

No importa cómo actualice la información del origen de datos, el cambio estará disponible para el servidor casi inmediatamente.

#### **Conceptos relacionados**

“Soporte Web” en la página 187

---

## **Script de construcción**

Un script de construcción es un archivo que es invocado por un plan de construcción y que prepara la salida de los archivos generados. A continuación se ofrecen algunos ejemplos:

- Un compilador Java u otro archivo .exe (binario) o un archivo .bat (texto) está disponible en un servidor de construcción del sistema de desarrollo o se envía a un servidor de construcción de un sistema Windows 2000/NT/XP remoto.
- Un script (archivo .scr) o una parte de código binario se envía a un servidor de construcción USS.

Especifique la dirección de una máquina de construcción estableciendo la opción del descriptor de construcción **destHost**.

## **Script de construcción Java**

Con el fin de preparar el código Java para su ejecución, EGL coloca el mandato javac (compilador Java) y sus parámetros en el plan de construcción y envía a la máquina de construcción el mandato javac y la entrada que necesita el mandato.

#### **Conceptos relacionados**

“Construcción” en la página 326

“Plan de construcción” en la página 327

“Servidor de construcción” en la página 343

#### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

“destDirectory” en la página 388

“destHost” en la página 388

“destPassword” en la página 389

“destUserID” en la página 390

“Salida de la generación de programa Java” en la página 675

“Salida de la generación de envoltura Java” en la página 676

---

## Servidor de construcción

Un servidor de construcción recibe peticiones de un sistema cliente para crear archivos ejecutables a partir del código fuente enviado desde dicho cliente. Se debe iniciar un servidor de construcción antes de enviar peticiones desde un cliente de construcción. Un servidor de construcción normalmente recibe peticiones de varios clientes. Se pueden iniciar varias hebras si se reciben peticiones de construcción concurrentes.

En un entorno de generador, un servidor de construcción se inicia en una máquina cuyo sistema operativo es el sistema de generación destino, como por ejemplo Windows 2000. El generador genera código fuente Java. Java se envía a un servidor de construcción especificado donde se invoca el compilador Java.

Si genera código Java para Windows, puede construir las salidas Java en la misma máquina que la máquina donde se ha realizado la generación. Esta construcción se denomina construcción local. En este caso, no es necesario iniciar un servidor de construcción. Si desea realizar una construcción local, omita la opción **destHost** del descriptor de construcción.

### Conceptos relacionados

“Construcción” en la página 326

“Script de construcción” en la página 342

### Tareas relacionadas

“Iniciar un servidor de construcción en AIX, Linux o Windows 2000/NT/XP”

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

## Iniciar un servidor de construcción en AIX, Linux o Windows 2000/NT/XP

Para iniciar un servidor de construcción remoto en AIX, Linux o Windows 2000/NT/XP, especifique el mandato `ccublds` en una ventana de indicador de mandatos. La sintaxis es la siguiente:

```
➔ ccublds — -p - númPuerto ————— ➔
```

```
 -V — [0] [2]
```

```
 -a — [0] [2]
```

donde

- p Especifica el número de puerto (*portno*) en el que el servidor está a la escucha para comunicarse con los clientes.
- V Especifica el nivel de verbosidad del servidor. Puede especificar este parámetro hasta tres veces (verbosidad máxima).
- a Especifica la modalidad de autenticación:
  - 0 El servidor realiza construcciones solicitadas por cualquier cliente. Esta modalidad se recomienda solamente en un entorno en el que la seguridad no es un problema.
  - 2 El servidor requiere que el cliente proporcione un ID de usuario y contraseña válidos antes de aceptar una construcción. El ID de usuario y la contraseña los configura primero el propietario del sistema

principal en el que se ejecuta el servidor de construcción. La configuración se efectúa utilizando el Gestor de seguridad descrito más abajo.

## Establecer el idioma de los mensajes devueltos desde el servidor de construcción

El servidor de construcción en Windows devuelve mensajes en cualquiera de los idiomas listados en la tabla siguiente y el valor por omisión es el Inglés.

| Idioma              | Código |
|---------------------|--------|
| Portugués de Brasil | ptb    |
| Chino simplificado  | chs    |
| Chino tradicional   | cht    |
| Inglés, EE.UU.      | enu    |
| Francés             | fra    |
| Alemán              | deu    |
| Italiano            | ita    |
| Japonés             | jpn    |
| Coreano             | kor    |
| Español             | esp    |

Para especificar un idioma que no sea el Inglés, asegúrese de que antes de iniciar el servidor de construcción, la variable de entorno `CCU_CATALOG` está establecida en un catálogo de mensajes no en Inglés. El valor necesario tiene el siguiente formato (en una sola línea):

```
dirInstalación\egl\eclipse\plugins
 \com.ibm.etools.egl.distributedbuild\executables
 \ccu.cat.xxx
```

### *dirInstalación*

El directorio de instalación del producto, como por ejemplo `C:\Program Files\IBM\RSPD\6.0`. Si instaló y tuvo un producto de Rational Developer antes de instalar el producto que está utilizando ahora, deberá especificar el directorio utilizado en la instalación anterior.

*xxx*

El código de idioma que el servidor de construcción soporta; uno de los códigos listados en la tabla anterior

## Gestor de seguridad

El Gestor de seguridad es un programa de servidor que el servidor de construcción utiliza para autenticar clientes que envían peticiones de construcción.

**Establecer el entorno para el Gestor de seguridad:** El Gestor de seguridad utiliza las siguientes variables de entorno de Windows:

### CCUSEC\_PORT

Establece el número del puerto en el que está a la escucha el Gestor de seguridad. El valor por omisión es 22825.

### CCUSEC\_CONFIG

Establece el nombre de vía de acceso del archivo en el que se guardan los datos de configuración. El valor por omisión es `C:\temp\ccuconfig.bin`. Si no se encuentra este archivo, el Gestor de seguridad lo crea.

## CCU\_TRACE

Inicia el rastreo del Gestor de seguridad con el fin de realizar diagnósticos, si esta variable está establecida en \*.

**Iniciar el Gestor de seguridad:** Para iniciar el Gestor de seguridad, emita el siguiente mandato:

```
java com.ibm.etools.egl.distributedbuild.security.CcuSecManager
```

**Configurar el Gestor de seguridad:** Para configurar el Gestor de seguridad, utilice la Herramienta de configuración, que tiene una interfaz gráfica. Puede ejecutar la herramienta emitiendo el siguiente mandato:

```
java com.ibm.etools.egl.distributedbuild.security.CCUconfig
```

Cuando la Herramienta de configuración esté ejecutándose, seleccione la pestaña **Elementos de servidor**. Utilizando el botón 'Añadir..', para añadir el usuario que desee que el servidor de construcción soporte, pulse en el botón **Añadir...** Debe definir una contraseña para el ID de usuario. Puede definir las siguientes restricciones y privilegios para el usuario:

- Las ubicaciones, es decir, los valores del parámetro `-la` del mandato `ccubldc` que este usuario puede especificar. Las distintas ubicaciones están separadas por punto y coma.
- El nombre del script de construcción que este usuario puede especificar. (El plan de construcción de EGL sólo utiliza el mandato `javac` como script de construcción.)
- Si este usuario puede o no puede enviar scripts de construcción desde el cliente, es decir, utilizar el parámetro `-ft` del mandato `ccubldc`. (El generador de EGL no utiliza el parámetro `-ft`. Especificaría este parámetro si se utilizara la construcción para fines que no fueran la preparación de salidas de generación de Java.)

Estas definiciones se conservan en almacenamiento persistente, en el archivo especificado por `CCUSEC_CONFIG` y se recuerdan en sesiones posteriores.

### Conceptos relacionados

“Script de construcción” en la página 342

“Servidor de construcción” en la página 343

### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755



---

# Desplegar salida Java generada por EGL

---

## Propiedades de tiempo de ejecución Java

Un programa Java generado por EGL utiliza un conjunto de propiedades de tiempo de ejecución que proporcionan información, como por ejemplo, cómo acceder a las bases de datos y archivos que utiliza el programa.

### En un entorno J2EE

En relación con un programa Java generado que se ejecutará en un entorno J2EE, se pueden dar las siguientes situaciones:

- EGL puede generar las propiedades de tiempo de ejecución directamente en un descriptor de despliegue J2EE. En este caso, EGL sobrescribe las propiedades que ya existen y añade las propiedades que no existen. El programa accede al descriptor de despliegue J2EE durante la ejecución.
- Como alternativa, EGL puede generar las propiedades de tiempo de ejecución en un archivo de entorno J2EE. Puede personalizar las propiedades en ese archivo y luego copiarlas en el descriptor de despliegue J2EE.
- Puede no generar ninguna propiedad de tiempo de ejecución, en cuyo caso debe escribir manualmente las propiedades necesarias.

En un módulo J2EE, cada programa tiene las mismas propiedades de tiempo de ejecución porque todo el código del módulo comparte el mismo descriptor de despliegue.

En WebSphere Application Server, las propiedades se especifican como códigos env-entry en el archivo web.xml asociado al proyecto Web, como en estos ejemplos:

```
<env-entry>
 <env-entry-name>vgj.nls.code</env-entry-name>
 <env-entry-value>ENU</env-entry-value>
 <env-entry-type>java.lang.String</env-entry-type>
</env-entry>

<env-entry>
 <env-entry-name>vgj.nls.number.decimal</env-entry-name>
 <env-entry-value>.</env-entry-value>
 <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

### En un entorno Java no J2EE

En relación con un programa Java generado que se ejecuta fuera de un entorno J2EE, puede generar las propiedades de tiempo de ejecución en un archivo de propiedades del programa o bien codificar manualmente dicho archivo. (El archivo de propiedades del programa proporciona el tipo de información que está disponible en el descriptor de despliegue, pero el formato de las propiedades es distinto).

En un entorno Java no J2EE, las propiedades pueden especificarse en varios archivos de propiedades; las búsquedas se realizan por este orden:

- **user.properties**
- Un archivo cuyo nombre es el siguiente:

*nombrePrograma.properties*

*nombrePrograma*

El primer programa de la unidad de ejecución

- **rununit.properties**

La utilización de **user.properties** es adecuada cuando especifica propiedades que son específicas de un usuario. EGL no genera contenido para este archivo.

La utilización de **rununit.properties** es especialmente apropiada cuando el primer programa de una unidad de ejecución no accede a un archivo o base de datos pero llama a programas que sí lo hacen:

- Al generar el llamador, puede generar un archivo de propiedades con un nombre para el programa, y el contenido podría no incluir propiedades relacionadas con el archivo o la base de datos
- Al generar el programa llamado, puede generar **rununit.properties** y el contenido estaría disponible para ambos programas

Ninguno de estos archivos es obligatorio y los programas simples no necesitan ninguno.

En tiempo de despliegue, se aplican las reglas siguientes:

- El archivo de propiedades de usuario (**user.properties**, si está presente) está en el directorio inicial del usuario, según viene determinado por la propiedad del sistema Java *user.home*.
  - La ubicación de un archivo de propiedades de programa (si está presente) depende de si el programa está en un paquete. Las reglas se ilustran mejor con ejemplos:
    - Si el programa P está en el paquete x.y.z y se despliega en MyProject/JavaSource, el archivo de propiedades del programa debe estar en MyProject/JavaSource/x/y/z
    - Si el programa P no está en un paquete y se despliega en myProject/JavaSource, el archivo de propiedades del programa (como el archivo de propiedades global) debe estar en MyProject/JavaSource
- En cualquier caso, MyProject/JavaSource debe estar en la vía de acceso de clases.
- El archivo de propiedades global (**rununit.properties**, si está presente) debe estar con el programa, en un directorio especificado en la vía de acceso de clases.

Si genera salida a un proyecto Java, sitúa los archivos de propiedades (que no sean **user.properties**) en las carpetas adecuadas.

Si está generando código Java para utilizarlo en la misma unidad de ejecución que el código Java generado con una versión anterior de EGL o VisualAge Generator, las reglas para desplegar archivos de propiedades dependen de si el primer programa de la unidad de ejecución se generó con EGL 6.0 o una versión posterior (en cuyo caso se aplican las reglas descritas aquí) o de si el primer programa se generó con una versión anterior de EGL o VisualAge Generator (en cuyo caso, los archivos de propiedades pueden estar en cualquier directorio de la vía de acceso de clases y el archivo global se llama **vgj.properties**).

Finalmente, si el primer programa se generó con el software más antiguo, puede especificar un archivo de propiedades alternativo que se utiliza en toda la unidad de ejecución en lugar de los archivos de propiedades de programa no globales.

Para obtener más detalles, consulte la descripción de la propiedad `vgj.properties.file` en las *propiedades de tiempo de ejecución Java (detalles)*.

## Descriptores de construcción y propiedades de programa

Las opciones se envían a EGL en forma de valores de opciones del descriptor de construcción:

- Para generar propiedades en un descriptor de despliegue J2EE, establezca **J2EE** en YES; establezca **genProperties** en PROGRAM o GLOBAL; y genere en un proyecto J2EE.
- Para generar propiedades en un archivo de entorno J2EE, establezca **J2EE** en YES; establezca **genProperties** en PROGRAM o GLOBAL; y realice una de estas dos acciones:
  - Genere en un directorio (en cuyo caso, utilice la opción del descriptor de construcción **genDirectory** en lugar de **genProject**); o bien
  - Genere en un proyecto no J2EE.
- Para generar un archivo de propiedades del programa con el mismo nombre que el programa que se está generando, establezca **J2EE** en NO; establezca **genProperties** en PROGRAM; y genere en un proyecto que no sea un proyecto J2EE.
- Para generar un archivo de propiedades del programa **rununit.properties**, establezca **J2EE** en NO; establezca **genProperties** en GLOBAL; y genere en un proyecto que no sea un proyecto J2EE.
- Para no generar propiedades, establezca **genProperties** en NO.

## Para obtener más información

Para obtener información detallada sobre cómo generar propiedades en un descriptor de despliegue o en un archivo de entorno J2EE, consulte la sección *Establecer valores del descriptor de despliegue*.

Para obtener información detallada sobre el significado de las propiedades de tiempo de ejecución, consulte la sección *Propiedades de tiempo de ejecución Java (detalles)*.

Para obtener detalles sobre cómo acceder a las propiedades de tiempo de ejecución en el código EGL, consulte la sección *sysLib.getProperty*.

### Conceptos relacionados

“Depurador EGL” en la página 279

“Generación de código Java en un proyecto” en la página 323

“Archivo de entorno J2EE” en la página 357

“Archivo de propiedades del programa” en la página 350

“Unidad de ejecución” en la página 742

### Tareas relacionadas

“Procesar código Java generado en un directorio” en la página 335

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

“Establecer valores de descriptor de despliegue” en la página 355

“Actualizar el descriptor de despliegue manualmente” en la página 357

“Actualizar el archivo de entorno J2EE” en la página 356

### Consulta relacionada

“genProperties” en la página 394

“J2EE” en la página 396

“Propiedades de ejecución de Java (detalles)” en la página 540

“getProperty()” en la página 903

---

## Configuración del entorno de tiempo de ejecución no J2EE para código generado por EGL

### Archivo de propiedades del programa

El *archivo de propiedades del programa* contiene propiedades de entorno de ejecución Java en un formato que sólo es accesible en un programa Java que se ejecuta fuera de un entorno J2EE. Para obtener información general, consulte la sección *Propiedades de entorno de ejecución Java*.

El archivo de propiedades del programa es un archivo de texto. Las entradas que no sean comentarios tienen el siguiente formato:

```
nombrePropiedad = valorPropiedad
```

*nombrePropiedad*

Una de las propiedades que se describen en la sección *Propiedades de entorno de ejecución Java (detalles)*

*propertyValue*

El valor de propiedad que está disponible en el programa durante la ejecución

Un comentario es cualquier línea donde el primer carácter no de texto es un signo #.

A continuación se ofrece una parte de un archivo de ejemplo:

```
Este archivo contiene propiedades para programas
Java generados que se están depurando en un
proyecto Java no J2EE
```

```
vgj.nls.code = ENU
vgj.datemask.gregorian.long.ENU = MM/dd/yyyy
```

Para obtener información detallada sobre el nombre asignado al archivo generado, consulte la sección *Salida generada (referencia)*.

#### Conceptos relacionados

“Depurador EGL” en la página 279

“Propiedades de tiempo de ejecución Java” en la página 347

#### Tareas relacionadas

“Salida generada (referencia)” en la página 530

#### Consulta relacionada

“genProperties” en la página 394

“J2EE” en la página 396

“Propiedades de ejecución de Java (detalles)” en la página 540

### Desplegar aplicaciones de Java fuera de J2EE

Para desplegar una aplicación de Java fuera de J2EE, haga lo siguiente:

1. Siga el procedimiento detallado en *Instalar el código de ejecución de EGL para Java*

2. Exporte el código generado por EGL en archivos jar, recordando incluir los archivos de salida generados que tengan extensiones de archivo que no sean java; por ejemplo, archivos jasper, properties, y tab
3. Exporte el código Java escrito manualmente a archivos jar
4. Incluya los archivos exportados en la vía de acceso de clases de la máquina destino

#### Tareas relacionadas

“Instalar el código de ejecución de EGL para Java”

## Instalar el código de ejecución de EGL para Java

El código de tiempo de ejecución EGL para las aplicaciones Java generadas está disponible en un archivo zip en el siguiente sitio Web:

<http://www3.software.ibm.com/ibmdl/pub/software/rationalsdp/rad/60/redist>

Las plataformas distribuidas soportadas son AIX, HP-UX, Linux (Intel), iSeries, Solaris y Windows 2000/NT/XP. (Consulte los requisitos previos del producto para conocer las versiones soportadas.) EGL suministra soporte de 32 y 64 bits para AIX, HP-UX y Solaris.

El archivo zip que bajará de los sitios Web mencionados anteriormente incluye lo siguiente:

- Archivos Jar que contienen código Java común a todas las plataformas distribuidas soportadas
- Código específico de plataforma

Haga lo siguiente:

1. Extraiga los archivos jar en el directorio EGLRuntimes de cada máquina en que deban ejecutarse aplicaciones EGL desplegadas fuera de un servidor de aplicaciones J2EE. (Estos archivos ya están incluidos en cualquier archivo Enterprise de archivado (EAR) utilizado para desplegar aplicaciones J2EE.)
2. Incluya los archivos jar en la vía de acceso de clases de las máquinas de despliegue
3. Copie el código específico de plataforma en un directorio de cada máquina de despliegue, y establezca variables de entorno para cada una de dichas máquinas según convenga:

#### Para AIX (soporte de 32 o 64 bits)

Los archivos de interés están el directorio **Aix** o (para el soporte de 64 bits) **Aix64**. Cambie las variables de entorno PATH y LIBPATH a fin de que hagan referencia al directorio que contiene el código específico de plataforma que ha copiado desde el sitio Web.

#### Para HP-UX (soporte de 32 o 64 bits)

Los archivos de interés están el directorio **hpux** o (para el soporte de 64 bits) **hpux64**. Cambie las variables de entorno PATH y LIBPATH a fin de que hagan referencia al directorio que contiene el código específico de plataforma que ha copiado desde el sitio Web.

#### Para iSeries

Los archivos de interés están el directorio **Iseries**. En qshell, cambie al directorio al que ha subido los archivos y ejecute el script setup.sh con la opción "install":

```
> setup.sh install
```

Además deben establecerse otras variables de entorno. Para obtener información sobre cómo establecer estas variables de entorno, ejecute el script con la opción "envinfo":

```
> setup.sh envinfo
```

Si por algún motivo suprime un symlink que se haya creado automáticamente durante la instalación, puede volver a crearlo con la opción "link":

```
> setup.sh link
```

#### Para Linux

Los archivos de interés están el directorio **Linux**. Cambie las variables de entorno PATH y LIBPATH a fin de que hagan referencia al directorio que contiene el código específico de plataforma que ha copiado desde el sitio Web.

#### Para Solaris (soporte de 32 o 64 bits)

Los archivos de interés están el directorio **Solaris** o (para el soporte de 64 bits) **Solaris64**. Cambie las variables de entorno PATH y LIBPATH a fin de que hagan referencia al directorio que contiene el código específico de plataforma que ha copiado desde el sitio Web.

#### Para Windows 2000/NT/XP

Los archivos de interés están el directorio **Win32**. Cambie la variable de entorno PATH a fin de que haga referencia al directorio que contiene el código específico de plataforma que ha copiado desde el sitio Web.

#### Tareas relacionadas

"Desplegar aplicaciones de Java fuera de J2EE" en la página 350

## Incluir archivos JAR en la variable CLASSPATH del sistema destino

Debe incluir los archivos JAR que contengan código generado por EGL o código Java escrito manualmente en la variable CLASSPATH del sistema destino. Los pasos para este proceso son dependientes del sistema. Consulte la documentación del sistema operativo para obtener detalles.

## Preparación de la biblioteca de curses UNIX curses para el entorno de ejecución de EGL

Cuando despliega un programa de texto EGL en AIX o Linux, el tiempo de ejecución EGL intenta utilizar la biblioteca curses UNIX. Si el entorno no está configurado para la biblioteca curses UNIX o si esa biblioteca no está soportada, el entorno de ejecución de EGL intenta utilizar la tecnología Swing Java; y si esa tecnología tampoco está disponible, el programa falla.

La biblioteca curses UNIX es necesaria cuando el usuario ejecuta un programa EGL desde una ventana de emulador de terminal o un terminal de caracteres.

Para habilitar el tiempo de ejecución de EGL para acceder a la biblioteca de terminal curses UNIX en AIX o Linux, debe seguir varios pasos en el entorno shell de UNIX. En cada uno de los dos primeros pasos, *installDir* hace referencia a la biblioteca de instalación del entorno de ejecución:

1. Modifique la variable de entorno LD\_LIBRARY\_PATH para que incluya el objeto compartido libCursesCanvas6.so, que se proporciona en la biblioteca de instalación del entorno de ejecución:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH: /installDir/bin
```

2. Modifique la variable de entorno CLASSPATH para añadir fda6.jar y fdaj6.jar:

```
export CLASSPATH=$CLASSPATH:
/installDir/lib/fda6.jar: /installDir/lib/fdaj6.jar
```

La información anterior debe teclearse en una sola línea.

3. Establezca la variable de entorno TERM en el valor de terminal adecuado, como en el ejemplo siguiente:

```
export TERM=vt100
```

Si se producen excepciones de terminal, pruebe varios valores de terminal como, por ejemplo, xterm, dterm o vt220.

4. Ejecute el programa Java EGL desde la shell UNIX, como en el ejemplo siguiente:

```
java myProgram
```

Asegúrese de que la variable de entorno CLASSPATH identifica el directorio en el que reside el programa.

Para obtener detalles adicionales sobre el uso de la biblioteca Curses en UNIX, consulte las páginas principales de UNIX.

## Configurar el escucha TCP/IP para una aplicación no J2EE llamada

Si desea que un llamante utilice TCP/IP para intercambiar datos con un programa Java no J2EE llamado, debe configurar un escucha TCP/IP para el programa llamado.

Si está utilizando TCP/IP para comunicarse con un programa Java no de J2EE llamado, debe configurar un programa Java autónomo denominado CSOTcpipListener para ese programa. Concretamente, debe hacer lo siguiente:

- Asegúrese de que la vía de acceso de clases utilizada al ejecutar CSOTcpipListener contiene fda6.jar, fdaj6.jar y los directorios o archivadores que contienen los programas llamados; y
- Establezca la propiedad de ejecución de Java **tcpiplistener.port** en el número del puerto en el que CSOTcpipListener recibe datos.

Puede iniciar el escucha TCP/IP autónomo de cualquiera de estas dos maneras:

- Para iniciar el escucha desde el entorno de trabajo, utilice la configuración de lanzamiento para una aplicación Java. En este caso, puede especificar el nombre del archivo de propiedades en los argumentos de programa de la configuración de lanzamiento. Otra posibilidad, si está utilizando el archivo tcpiplistener.properties como archivo por omisión, ese archivo no deberá estar en una carpeta, sino directamente bajo el proyecto que ha especificado al crear la configuración de lanzamiento.
- Para iniciar el escucha desde la línea de mandatos, ejecute el programa como se indica a continuación:

```
java CSOTcpipListener propertiesFile
```

```
propertiesFile
```

La vía de acceso totalmente calificado al archivo de propiedades utilizado por el escucha de TCP/IP. Si no especifica un archivo de propiedades, el escucha intenta abrir el siguiente archivo en el directorio actual:

```
tcpiplistener.properties
```

### Tareas relacionadas

“Proporcionar acceso a archivos jar no de EGL” en la página 365

---

## Configurar entorno de ejecución J2EE para código generado por EGL

Los programas y envolturas Java generados por EGL se ejecutan en un servidor J2EE 1.4 como por ejemplo WebSphere Application Server v6.0, en las plataformas indicadas en *Configuraciones de ejecución*.

Las tareas primarias al intercalar clases de Java generadas en un módulo J2EE son las siguientes:

1. Colocar archivos de salida en un proyecto, de una de dos maneras:
  - Generar en un proyecto, que es la técnica preferida; o bien
  - Generar en un directorio y, a continuación, importar archivos a un proyecto.
2. Colocar un archivo de propiedades de enlace en el módulo (consulte *Desplegar un archivo de propiedades de enlace*).
3. Eliminar archivo jar duplicados.
4. Exportar un archivo Enterprise de archivado (.ear), que puede incluir archivos de archivado de aplicación Web (.war) y otros archivos .ear; para conocer detalles sobre el procedimiento, consulte las páginas de ayuda sobre la exportación.
5. Importe el archivo .ear en el servidor J2EE que albergará la aplicación; para conocer detalles sobre el procedimiento, consulte la documentación del servidor J2EE.

Puede ser necesario cumplimentar también estas tareas:

- “Establecer una conexión JDBC J2EE” en la página 362
- “Configuración del servidor J2EE para llamadas CICSJ2C” en la página 358
- “Config. escucha TCP/IP para aplic. llamada en módulo cliente aplic. J2EE” en la página 359
- “Configurar el escucha TCP/IP para una aplicación no J2EE llamada” en la página 353

### Conceptos relacionados

“Proceso de desarrollo” en la página 8

“Generación de código Java en un proyecto” en la página 323

“Programa Java, PageHandler y biblioteca” en la página 328

“Componente de opciones de enlace” en la página 311

“Archivo de propiedades de enlace” en la página 364

“Configuraciones de tiempo de ejecución” en la página 9

### Tareas relacionadas

“Desplegar aplicaciones de Java fuera de J2EE” en la página 350

“Desplegar un archivo de propiedades de enlace” en la página 364

“Eliminar archivos jar duplicados” en la página 355

“Generar código de despliegue para proyectos EJB” en la página 338

“Procesar código Java generado en un directorio” en la página 335

“Proporcionar acceso a archivos jar no de EGL” en la página 365

“Establecer valores de descriptor de despliegue” en la página 355

“Establecer el nombre JNDI para proyectos EJB” en la página 358

“Establecer una conexión JDBC J2EE” en la página 362

“Configuración del servidor J2EE para llamadas CICSJ2C” en la página 358

“Config. escucha TCP/IP para aplic. llamada en módulo cliente aplic. J2EE” en la página 359

- “Cómo se realiza una conexión JDBC estándar” en la página 263
- “Actualizar el descriptor de despliegue manualmente” en la página 357
- “Actualizar el archivo de entorno J2EE” en la página 356

**Consulta relacionada**

- “Propiedades de ejecución de Java (detalles)” en la página 540
- “Archivo de propiedades de enlace (detalles)” en la página 657

## Eliminar archivos jar duplicados

Si coloca múltiples módulos J2EE en un único archivo ear, elimine los archivos jar duplicados como se indica a continuación:

1. Mueva una copia de cada archivo jar duplicado al nivel superior del ear
2. Suprima los archivos jar duplicados de los módulos J2EE
3. Asegúrese de que la vía de acceso de construcción para cada uno de los módulos J2EE afectados señala a los archivos jar del ear; concretamente, haga lo siguiente para cada uno de esos módulos J2EE:
  - a. Pulse con el botón derecho del ratón en el módulo desde dentro de la vista Explorador de proyectos o la vista J2EE
  - b. Seleccione **Editar dependencias de módulo**
  - c. Cuando se visualice el diálogo Dependencias de módulo, seleccione los archivos jar para acceder desde el nivel superior del ear y, a continuación, pulse en **Finalizar**.

**Tareas relacionadas**

- “Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

## Establecer valores de descriptor de despliegue

Una tarea importante es colocar valores de ejecución (similar a valores de variables de entorno) en el descriptor de despliegue del módulo J2EE. Puede interactuar con un editor del entorno de trabajo listado en la tabla siguiente, por ejemplo; y en cualquier caso, los editores están disponibles si desea reasignar un valor.

Tipo de proyecto	Nombre del descriptor de despliegue	Cómo asignar valores
cliente de aplicaciones	application-client.xml	Utilice el editor XML, pestaña Diseño
EJB	ejb-jar.xml	Utilice el editor EJB, pestaña Beans
Web J2EE	web.xml	Utilice el editor web.xml, pestaña Entorno

La manera recomendada de actualizar el descriptor de despliegue es añadir contenido automáticamente, como sucede si se cumplen todas las condiciones siguientes:

- Está generando un programa o envoltura Java
- La opción del descriptor de construcción **genProperties** está establecida en GLOBAL o PROGRAM
- Está generando para la ejecución de J2EE estableciendo **J2EE** en YES
- Establece **genProject** como un proyecto J2EE válido

EGL nunca suprime una propiedad de un descriptor de despliegue existente, pero hace lo siguiente:

- Escribe encima de las propiedades que ya existen
- Añade propiedades que no existen

Otro método de actualizar el descriptor de despliegue es pegar valores desde el archivo de entorno J2EE, que es una salida de generación si se cumplen todas las condiciones siguientes:

- Está generando un programa Java
- La opción del descriptor de construcción **genProperties** está establecida en GLOBAL o PROGRAM
- Está generando para la ejecución de J2EE estableciendo **J2EE** en YES
- No establece **genProject** como un proyecto J2EE válido, como cuando genera en un directorio

Antes de pegar entradas desde un archivo de entorno J2EE al descriptor de despliegue de un cliente de aplicaciones o proyecto EJB, deberá cambiar el orden de las entradas en el archivo, tal como se describe en *Actualizar el archivo de entorno J2EE*. No es necesario cambiar el orden de las entradas si está trabajando con un proyecto Web J2EE.

Encontrará los detalles sobre las propiedades de los descriptores de despliegue en *Propiedades de tiempo de ejecución de Java (detalles)*.

#### **Conceptos relacionados**

“Archivo de entorno J2EE” en la página 357

“Generación de código Java en un proyecto” en la página 323

“Archivo de propiedades del programa” en la página 350

#### **Tareas relacionadas**

“Procesar código Java generado en un directorio” en la página 335

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

“Actualizar el archivo de entorno J2EE”

“Actualizar el descriptor de despliegue manualmente” en la página 357

#### **Consulta relacionada**

“genDirectory” en la página 391

“genProperties” en la página 394

“J2EE” en la página 396

“Propiedades de ejecución de Java (detalles)” en la página 540

## **Actualizar el archivo de entorno J2EE**

El archivo de entorno J2EE contiene una serie de entradas como las del siguiente ejemplo:

```
<env-entry>
 <env-entry-name>vgj.nls.code</env-entry-name>
 <env-entry-value>ENU</env-entry-value>
 <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

El orden de subelementos es nombre, valor, tipo. Esto es correcto para proyectos Web J2EE; no obstante, para los proyectos de cliente de aplicaciones y EJB, debe cambiar el orden a nombre, tipo, valor. Para el ejemplo anterior, cambie el orden de los subelementos a:

```
<env-entry>
 <env-entry-name>vgj.nls.code</env-entry-name>
 <env-entry-type>java.lang.String</env-entry-type>
 <env-entry-value>ENU</env-entry-value>
</env-entry>
```

Este paso puede evitarse si genera directamente en un proyecto en lugar de en un directorio. Al generar en un proyecto, EGL puede determinar el tipo de proyecto que está utilizando y generar las entradas de entorno en el orden adecuado.

#### **Tareas relacionadas**

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354  
“Establecer valores de descriptor de despliegue” en la página 355

#### **Consulta relacionada**

“Propiedades de ejecución de Java (detalles)” en la página 540

### **Archivo de entorno J2EE**

Un *archivo de entorno J2EE* es un archivo de texto que contiene pares propiedad-valor que se derivan de la información especificada cuando se genera un programa Java. Las fuentes de información son el descriptor de construcción, el componente de asociaciones de recursos y el componente de opciones de enlace.

Cuando configure el entorno del programa Java, puede utilizar el archivo de entorno J2EE como base de la información que se coloca en el descriptor de despliegue en tiempo de ejecución.

Para obtener información detallada sobre el nombre del archivo de entorno J2EE, consulte la sección *Salida generada (referencia)*.

Para obtener información detallada sobre las diferentes formas en que puede establecer valores del descriptor de despliegue, consulte la sección *Establecer valores del descriptor de despliegue*.

#### **Conceptos relacionados**

“Configuraciones de tiempo de ejecución” en la página 9

#### **Tareas relacionadas**

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354  
“Establecer valores de descriptor de despliegue” en la página 355

#### **Consulta relacionada**

“Salida generada (referencia)” en la página 530  
“genProperties” en la página 394  
“sqlDB” en la página 400

## **Actualizar el descriptor de despliegue manualmente**

Si está actualizando un descriptor de despliegue desde un archivo de entorno J2EE generado, haga lo siguiente:

1. Lea la información de visión general en *Establecer valores de descriptor de despliegue*.
2. Si ha trabajado en un cliente de aplicaciones o en un proyecto EJB, debe asegurarse de que el orden de los subelementos en las entradas de entorno generadas es correcto, tal como se describe en *Actualizar el archivo de entorno J2EE*.

3. Copie las entradas de entorno en el descriptor de despliegue del proyecto como se indica a continuación:
  - a. Haga una copia de seguridad del descriptor de despliegue.
  - b. Abra el archivo de entorno de J2EE, que se denomina archivo *programName-env.txt*. Copie las entradas de entorno en el área común.
  - c. Realice una doble pulsación en el descriptor de despliegue.
  - d. Pulse en la pestaña Fuente.
  - e. Pegue las entradas en la ubicación correcta.

Para conocer más detalles acerca de los descriptores de despliegue, consulte el apartado *Propiedades de tiempo de ejecución Java (detalles)*.

#### **Tareas relacionadas**

“Establecer valores de descriptor de despliegue” en la página 355

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

“Actualizar el archivo de entorno J2EE” en la página 356

#### **Consulta relacionada**

“Propiedades de ejecución de Java (detalles)” en la página 540

## **Establecer el nombre JNDI para proyectos EJB**

Para establecer el nombre JNDI para un proyecto EJB, haga lo siguiente:

1. Pulse con el botón derecho del ratón en *ejb-jar.xml* (el descriptor de despliegue) para abrir el menú de contexto.
2. Utilice el Editor de EJB para abrir el siguiente archivo del proyecto:  
`\ejbModule\META-INF\ejb-jar.xml`
3. Pulse en la pestaña Beans.
4. En la lista, pulse en el nombre del EJB que acaba de generar.
5. Entre el nombre JNDI bajo Enlaces WebSphere. El nombre JNDI debe ser como se indica a continuación para que lo utilice el código de tiempo de ejecución de EGL:
  - El primer carácter del nombre de programa, en mayúsculas
  - Los caracteres siguientes del nombre de programa, en minúsculas
  - Las letras EJB en mayúsculas.

#### **Tareas relacionadas**

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

## **Configuración del servidor J2EE para llamadas CICSJ2C**

Debe configurar una *ConnectionFactory* en el servidor J2EE para cada transacción CICS a la que se accede mediante el protocolo CICSJ2C.

Si una envoltura Java generada está realizando la llamada CICSJ2C, puede gestionar la seguridad de cualquiera de las siguientes maneras (donde un valor especificado por la envoltura prevalece sobre el del servidor J2EE):

- Establezca el ID de usuario y contraseña en el objeto *CSOCallOptions* de la envoltura; o bien
- Establezca el ID de usuario y contraseña en la configuración de *ConnectionFactory* en el servidor J2EE; o bien
- Configure la región de CICS de forma que no sea necesaria la autenticación de usuario.

Al llamar a un programa desde WebSphere 390, corresponden las siguientes restricciones:

- Si la propiedad del elemento callLink **luwControl** está establecida en CLIENT, la llamada falla. La implementación de conexión de WebSphere 390 no da soporte a una unidad de trabajo extendida.
- El valor de la propiedad del descriptor de despliegue **cso.cicsj2c.timeout** no tiene efecto alguno.

Por omisión no se exceden los tiempos de espera. No obstante, en la tabla de opciones de EXCI generada por la macro DFHXCOPT, puede establecer el parámetro TIMEOUT, que le permite especificar el tiempo que EXCI esperará a que un mandato DPL (una petición ECI) se complete. Un valor 0 significa esperar indefinidamente.

Encontrará los detalles en *Java Connectors for CICS: Featuring the J2EE Connector Architecture* (SG24-6401-00), que está disponible en el sitio web <http://www.redbooks.ibm.com>.

#### Tareas relacionadas

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

## Config. escucha TCP/IP para aplic. llamada en módulo cliente aplic. J2EE

Si desea que un llamante utilice TCP/IP para intercambiar datos con un programa llamado en un módulo de cliente de aplicaciones J2EE, debe configurar un escucha TCP/IP para el programa llamado.

Debe asegurarse de que la situación siguiente esté en vigor:

- Un escucha de TCP/IP específico para EGL es la clase principal para el módulo, como se especifica en el archivo de manifiesto (.MF) del módulo
- Se ha asignado un puerto al escucha, como se especifica en el descriptor de despliegue (application-client.xml) del módulo

Si está trabajando con proyectos en el nivel de J2EE 1.2, es recomendable configurar un proyecto de cliente de aplicaciones que se inicialice con el escucha, antes de generar código EGL en ese proyecto. Si no consigue seguir esa secuencia (escucha primero, código EGL segundo) o si está trabajando con proyectos en el nivel de J2EE 1.3, deberá seguir el procedimiento descrito en *Proporcionar acceso al escucha desde un proyecto de cliente de aplicaciones existente*.

### Configuración de un proyecto de cliente de aplicaciones que se inicialice con el escucha

Para configurar un proyecto de cliente de aplicaciones que se inicialice con el escucha, haga lo siguiente:

1. Pulse **Archivo > Importar**.
2. En la página Seleccionar, pulse dos veces en **Archivo JAR de cliente apl.**
3. En la página Importación de cliente de aplicaciones, especifique varios detalles.
  - a. En el campo Archivo de cliente de aplicaciones, especifique el archivo jar que configura el acceso (pero que no incluye) el escucha de TCP/IP:

```
dirInstalación\egl\eclipse\plugins\
com.ibm.etools.egl.generators_versión\runtime\EGLTcpiListener.jar
```

*dirInstalación*

El directorio de instalación del producto, como por ejemplo C:\Program

Files\IBM\RSPD\6.0. Si instaló y tuvo un producto de Rational Developer antes de instalar el producto que está utilizando ahora, deberá especificar el directorio utilizado en la instalación anterior.

*versión*

La versión más reciente del conector; por ejemplo, 6.0.0.

El propio escucha de TCP/IP reside en fdaj6.jar, que se coloca en el proyecto de cliente de aplicaciones al generar por primera vez código EGL en ese proyecto.

- b. Pulse el botón de selección **Nuevo**, que está a continuación de la etiqueta **Proyecto de cliente de aplicaciones**.
- c. Teclee el nombre del proyecto de cliente de aplicaciones en el campo **Nombre de proyecto nuevo**; a continuación seleccione o deseleccione el recuadro de selección **Utilizar valor por omisión**. Si marca el recuadro de selección, el proyecto se almacena en un directorio del espacio de trabajo denominado con el nombre del proyecto. Si quita la marca del recuadro de selección, especifique el nombre de proyecto en el campo **Ubicación de proyecto nuevo**.
- d. Especifique el nombre del proyecto de aplicación de empresa que contiene el proyecto de cliente de aplicaciones:
  - Si está utilizando un proyecto de aplicación de empresa J2EE 1.2 existente, pulse en el botón de selección **Existente**, que está a continuación de la etiqueta **Proyecto de aplicación de empresa**. En este caso, especifique el nombre de proyecto en el campo **Nombre de proyecto existente**.
  - Si está creando un proyecto de aplicación de empresa nuevo, haga lo siguiente:
    - 1) Pulse el botón de selección **Nuevo**, que está a continuación de la etiqueta **Proyecto de aplicación de empresa**.
    - 2) Teclee el nombre del proyecto de aplicación de empresa en el campo **Nombre de proyecto nuevo**.
    - 3) Seleccione o deseleccione el recuadro de selección **Utilizar valor por omisión**.
    - 4) Si marca el recuadro de selección, el proyecto se almacena en un directorio del espacio de trabajo denominado con el nombre del proyecto. Si quita la marca del recuadro de selección, especifique el nombre de proyecto en el campo **Ubicación de proyecto nuevo**.
4. Pulse en **Finalizar**.
5. Ignore los dos mensajes de aviso que hacen referencia a los archivos jar (fda6.jar, fdaj6.jar) que se añadirán automáticamente al generar salida de EGL en el proyecto.

En el proyecto de cliente de aplicaciones, la propiedad del descriptor de despliegue **tcpiplistener.port** se establece en el número del puerto en el que el escucha recibe datos. Por omisión, ese número de puerto es 9876. Para cambiar el número de puerto, haga lo siguiente:

1. En la vista Explorador de proyectos, expanda el proyecto de cliente de aplicaciones, a continuación appClientModule y seguidamente META-INF
2. Pulse en **application-client.xml > Abrir con > Editor del descriptor de despliegue**
3. El editor del descriptor de despliegue incluye una pestaña fuente; pulse esa pestaña y cambie el valor 9876, que es el contenido del último código en una agrupación como la siguiente:

```
<env-entry-name>tcpipListener.port</env-entry-name>
<env-entry-type>java.lang.Integer</env-entry-name>
<env-entry-value>9876</env-entry-value>
```

4. Para guardar el descriptor de despliegue, pulse **Control-S**.

## Proporcionar acceso al escucha desde un proyecto de cliente de aplicaciones existente

Si genera código EGL en un proyecto de cliente de aplicaciones que se ha inicializado con el escucha, deberá actualizar el descriptor de despliegue (application-client.xml) y el archivo de manifiesto (MANIFEST.MF):

1. En la vista Explorador de proyectos, expanda el proyecto de cliente de aplicaciones, a continuación appClientModule y seguidamente META-INF
2. Pulse en **application-client.xml > Abrir con > Editor del descriptor de despliegue**
3. El editor del descriptor de despliegue incluye una pestaña Fuente. Pulse en esa pestaña. En el texto, justo debajo de la línea que contiene el código `<display-name>`, añada las siguientes entradas (no obstante, si el puerto 9876 de la máquina ya se está utilizando, sustituya 9876 por un número distinto):

```
<env-entry>
 <env-entry-name>tcpipListener.port</env-entry-name>
 <env-entry-type>java.lang.Integer</env-entry-name>
 <env-entry-value>9876</env-entry-value>
</env-entry>
```
4. Para guardar el descriptor de despliegue, pulse **Control-S**.
5. En la vista Explorador de proyectos, pulse **MANIFEST.MF > Abrir con > Editor de Dependencias JAR**.
6. El Editor de Dependencias JAR incluye una pestaña Dependencias. Pulse en esa pestaña.
7. Revise la sección de Dependencias para asegurarse de que se han seleccionado fda6.jar y fdaj6.jar.
8. En la sección Clase principal, en el campo Clase principal, teclee el siguiente valor o utilice el mecanismo de Examinar para especificar el siguiente valor:  
CS0TcpipListenerJ2EE
9. Para guardar el archivo de manifiesto, pulse **Control-S**.

## Desarrollar el proyecto de cliente de aplicaciones

Para iniciar el escucha de TCP/IP, siga cualquiera de los dos procedimientos:

- Inicie el escucha desde el Entorno de trabajo utilizando la configuración de lanzamiento para un cliente de aplicaciones de WebSphere:
  1. Conmute a una perspectiva J2EE
  2. Pulse **Ejecutar > Ejecutar**
  3. En la página Configuraciones de lanzamiento, pulse en **Cliente de aplicaciones WebSphere v5** (necesario si está trabajando con un proyecto en el nivel de J2EE 1.3) o **Cliente de aplicaciones de WebSphere v4**
  4. Seleccione una configuración existente. Otra posibilidad es pulsar en **Nuevo** y preparar una configuración:
    - a. En la pestaña Aplicación, seleccione el proyecto de aplicación de empresa
    - b. En la pestaña Argumentos, añada un argumento:  
`-CCjar=myJar.jar`  
*myJar.jar*

El nombre del archivo jar del cliente de aplicaciones. Este argumento solamente es necesario cuando tiene múltiples archivos jar de cliente

en el archivo ear. En la mayoría de casos, el valor es el nombre del proyecto de cliente de aplicaciones, seguido de la extensión .jar.

Si desea confirmar la relación de nombre de proyecto a nombre de archivo jar, haga lo siguiente:

- 1) En la vista Explorador de proyectos, expanda el proyecto de aplicación de empresa y, a continuación, META-INF
  - 2) Pulse en **application.xml > Abrir con > Editor del descriptor de despliegue.**
  - 3) El Editor del descriptor de despliegue incluye una pestaña Módulo. Pulse en esa pestaña.
  - 4) En la parte izquierda de la página, pulse en el archivo jar y vea (en la parte derecha de la página) el nombre de proyecto asociado con ese archivo jar.
- Si tiene instalado WebSphere Application Server (WAS), puede utilizar launchClient.bat, que está en el directorio de instalación de WAS, subdirectorio bin.

Puede invocar launchClient desde un indicador de mandatos como se indica a continuación:

```
launchClient myCode.ear -CCjar=myJar.jar
```

*myCode.ear*

El nombre del Enterprise de archivado

*myJar.jar*

El nombre del archivo jar del cliente de aplicaciones, como se describe en relación con el procedimiento del entorno de trabajo

Para conocer detalles sobre launchClient.bat, consulte la documentación de WebSphere Application Server.

#### Tareas relacionadas

“Proporcionar acceso a archivos jar no de EGL” en la página 365

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

## Establecer una conexión JDBC J2EE

Si está conectándose a una base de datos relacional en tiempo de ejecución, debe definir un origen de datos para utilizarlo con el programa. Las indicaciones están en el sistema de ayuda de la consola administrativa del servidor WebSphere.

Al definir un origen de datos, asigne valores a las siguientes propiedades:

#### Nombre JNDI

Especifique un valor que coincida con el nombre al que está enlazada la base de datos en el registro de JNDI:

- Si está definiendo un origen de datos que se conecta a una base de datos que el módulo J2EE utiliza por omisión, asegúrese de que el nombre JNDI especificado en la definición de origen de datos coincida con el valor de la propiedad **vgj.jdbc.default.database** en el descriptor de despliegue J2EE utilizado en la ejecución
- Si está definiendo un origen de datos al que se acceda cuando se ejecute la función del sistema VGLib.connectionService, asegúrese de que el nombre JNDI especificado en la definición del origen de datos coincida con el valor de la propiedad **vgj.jdbc.database.SN** adecuada en el descriptor de despliegue J2EE utilizado durante la ejecución

### Nombre de base de datos

Especifique el nombre de la base de datos, como la conozca el sistema de gestión de bases de datos

### ID de usuario

Especifique el nombre de usuario para conectarse a la base de datos.

Si la definición de origen de datos hace referencia a la base de datos por omisión, el valor que especifique en el campo ID de usuario quedará alterado temporalmente por cualquier valor establecido en la propiedad

**vgj.jdbc.default.userid** del descriptor de despliegue J2EE utilizado durante la ejecución, pero solamente si ha especificado valores para

**vgj.jdbc.default.userid** y **vgj.jdbc.default.password**. De forma similar, si la definición de origen de datos hace referencia a una base de datos a la que se accede mediante la función del sistema `sysLib.connect` o

`VGLib.connectionService`, el valor que especifique en el campo ID de usuario queda alterado temporalmente por cualquier ID de usuario que especifique en la llamada a esa función del sistema, pero solamente si la llamada pasa un ID de usuario y una contraseña.

El nombre se especifica al configurar el alias de autenticación. Para alcanzar la pantalla en la que puede definir ese alias, siga esta secuencia en la Consola administrativa: **Seguridad > GlobalSecurity > Autenticación > Configuración JAAS > Datos de autenticación J2C**.

### Contraseña

Especifique la contraseña para conectarse a la base de datos. Si la definición de origen de datos hace referencia a la base de datos por omisión, el valor que especifique en el campo Contraseña quedará alterado temporalmente por cualquier valor establecido en la propiedad **vgj.jdbc.default.password** del descriptor de despliegue J2EE utilizado durante la ejecución, pero solamente si ha especificado valores para **vgj.jdbc.default.userid** y **vgj.jdbc.default.password**. De forma similar, si la definición de origen de datos hace referencia a una base de datos a la que se accede mediante la función del sistema `VGLib.connectionService`, el valor que especifique en el campo Contraseña queda alterado temporalmente por cualquier contraseña que especifique en la llamada a esa función del sistema, pero solamente si la llamada pasa un ID de usuario y una contraseña.

La contraseña se especifica al configurar el alias de autenticación. Para alcanzar la pantalla en la que puede definir ese alias, siga esta secuencia en la Consola administrativa: **Seguridad > GlobalSecurity > Autenticación > Configuración JAAS > Datos de autenticación J2C**.

Puede definir múltiples orígenes de datos, en cuyo caso utilizará la función del sistema `VGLib.connectionService` para conmutar entre ellos.

Para conocer detalles sobre el significado de las propiedades del descriptor de despliegue, incluidos detalles sobre cómo se derivan los valores generados, consulte *Propiedades de tiempo de ejecución de Java (referencia)*.

### Tareas relacionadas

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

“Cómo se realiza una conexión JDBC estándar” en la página 263

### Consulta relacionada

“Propiedades de ejecución de Java (detalles)” en la página 540

“Requisitos de controlador JDBC en EGL” en la página 560

“connectionService()” en la página 916

## Desplegar un archivo de propiedades de enlace

El archivo de propiedades de enlace debe estar en la misma aplicación J2EE que el programa Java que utiliza el archivo. Si el archivo está en el directorio de nivel superior de la aplicación, establezca la propiedad de tiempo de ejecución Java **cs.linkageOptions.LO** en el nombre del archivo, sin información de vía de acceso. Si el archivo está bajo el directorio de nivel superior de la aplicación, utilice una vía de acceso que empiece en el directorio de nivel superior e incluya una barra inclinada (/) para cada nivel, incluso aunque la aplicación esté ejecutándose en una plataforma Windows.

Cuando desarrolla un proyecto J2EE, el directorio de nivel superior corresponde a appClientModule, ejbModule, o el directorio de Contenido Web del proyecto en el que reside el módulo. Cuando desarrolla un proyecto Java, el directorio de nivel superior es el directorio del proyecto.

Para conocer detalles adicionales sobre el formato de un archivo de propiedades de enlace y su identificación, consulte *Archivo de propiedades de enlace (referencia)*.

### Conceptos relacionados

“Propiedades de tiempo de ejecución Java” en la página 347

“Componente de opciones de enlace” en la página 311

“Archivo de propiedades de enlace”

### Tareas relacionadas

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

“Establecer valores de descriptor de despliegue” en la página 355

### Consulta relacionada

“Elemento callLink” en la página 407

“Manejo de excepciones” en la página 94

“Archivo de propiedades de enlace (detalles)” en la página 657

## Archivo de propiedades de enlace

Un *archivo de propiedades de enlace* es un archivo de texto que se utiliza durante la ejecución Java para proporcionar detalles sobre cómo una envoltura o programa Java generado llama a un programa Java generado de otro proceso.

El archivo sólo es aplicable si ha especificado que las opciones de enlace para un programa o envoltura Java se establezcan durante la ejecución y no durante la generación. Puede generar el archivo o crear uno desde cero.

Para obtener información detallada sobre cuándo se genera el archivo y sobre el formato del archivo, consulte la sección *Archivo de propiedades de enlace (detalles)*. Para obtener información detallada sobre el nombre del archivo generado, consulte la sección *Salida generada (referencia)*. Para obtener información detallada sobre el despliegue, consulte la sección *Desplegar un archivo de propiedades de enlace*.

### Conceptos relacionados

“Salida generada” en la página 529

### Tareas relacionadas

“Desplegar un archivo de propiedades de enlace” en la página 364

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

### Consulta relacionada

“Salida generada (referencia)” en la página 530

“genProperties” en la página 394

“Archivo de propiedades de enlace (detalles)” en la página 657

## Proporcionar acceso a archivos jar no de EGL

Puede ser necesario proporcionar acceso a archivos jar no de EGL para depurar y ejecutar su código Java generado por EGL. El proceso para proporcionar acceso a esos archivos varía según el tipo de proyecto:

### Proyecto de cliente de aplicaciones

Antes de utilizar el depurador interpretativo, referencie los archivos jar no de EGL en la variable CLASSPATH, tal como se describe en *Establecer preferencias para el depurador de EGL*.

Antes de ejecutar el código (con o sin el depurador Java de EGL), haga lo siguiente:

1. Para cada proyecto de aplicación de empresa que haga referencia al proyecto de cliente de aplicaciones, importe archivos jar de interés desde un directorio del sistema de archivos:
  - a. En la vista Explorador de proyectos, pulse con el botón derecho del ratón en un proyecto de aplicación de empresa y pulse en **Importar**
  - b. En la página Seleccionar, pulse en **Sistema de archivos**
  - c. En la página Sistema de archivos, especifique el directorio en el que residen los archivos jar
  - d. En la parte derecha de la página, seleccione los archivos jar de interés
  - e. Pulse en **Finalizar**
2. Actualice el manifiesto en el proyecto de cliente de aplicaciones de forma que los archivos jar del proyecto de aplicación de empresa estén disponibles durante la ejecución:
  - a. En la vista Explorador de proyectos, pulse con el botón derecho del ratón en el proyecto de cliente de aplicaciones y pulse en **Propiedades**
  - b. A la izquierda de la página Propiedades, pulse en **Dependencias de JAR Java**
  - c. Cuando aparezca la sección denominada Dependencias JAR Java a la derecha de la página, marque cada recuadro de selección que corresponda a un archivo jar de interés
  - d. Pulse **Aceptar**

### Proyecto EJB

Antes de utilizar el depurador interpretativo, referencie los archivos jar no de EGL en la variable CLASSPATH, tal como se describe en *Establecer preferencias para el depurador de EGL*.

Antes de ejecutar el código (con o sin el depurador Java de EGL), haga lo siguiente:

1. Para cada proyecto de aplicación de empresa que haga referencia al proyecto EJB, importe los archivos jar de interés desde un directorio del sistema de archivos:

- a. En la vista Explorador de proyectos, pulse con el botón derecho del ratón en un proyecto de aplicación de empresa y pulse en **Importar**
  - b. En la página Seleccionar, pulse en **Sistema de archivos**
  - c. En la página Sistema de archivos, especifique el directorio en el que residen los archivos jar
  - d. En la parte derecha de la página, seleccione los archivos jar de interés
  - e. Pulse en **Finalizar**
2. Actualice el manifiesto en el proyecto EJB de forma que los archivos jar del proyecto de aplicación de empresa estén disponibles durante la ejecución:
    - a. En la vista Explorador de proyectos, pulse con el botón derecho del ratón en el proyecto EJB y pulse en **Propiedades**
    - b. A la izquierda de la página Propiedades, pulse en **Dependencias de JAR Java**
    - c. Cuando aparezca la sección denominada Dependencias JAR Java a la derecha de la página, marque cada recuadro de selección que corresponda a un archivo jar de interés
    - d. Pulse **Aceptar**

### Proyecto Java

Antes de ejecutar el código con el depurador interpretativo, referencie los archivos jar no de EGL en la variable CLASSPATH, tal como se describe en *Establecer preferencias para el depurador de EGL*.

Antes de ejecutar el código con el depurador Java EGL, añada entradas a la vía de construcción Java del proyecto:

1. En la vista Explorador de proyectos, pulse con el botón derecho del ratón en el proyecto Java y pulse en **Propiedades**
2. A la izquierda de la página Propiedades, pulse en **Vía de construcción Java**
3. Cuando aparezca la sección denominada Vía de construcción Java a la derecha de la página, pulse en la pestaña Bibliotecas
4. Para cada archivo jar a añadir, pulse en **Añadir jar externos** y utilice el mecanismo de Examinar para seleccionar el archivo
5. Para cerrar la página Propiedades, pulse en **Aceptar**

### Proyecto Web J2EE

Antes de utilizar el depurador interpretativo, referencie los archivos jar no de EGL en la variable CLASSPATH, tal como se describe en *Establecer preferencias para el depurador de EGL*.

Antes de ejecutar el código (con o sin el depurador Java EGL), importe los archivos jar desde el sistema de archivos a la siguiente carpeta de proyecto Web:

Web Content/WEB-INF/lib

El proceso de importación es el siguiente para un conjunto de archivos jar en un directorio:

1. En la vista Explorador de proyectos, expanda el proyecto Web, expanda **Web Content**, expanda **WEB-INF**, pulse con el botón derecho del ratón en **lib** y pulse en **Importar**
2. En la página Seleccionar, pulse en **Sistema de archivos**
3. En la página Sistema de archivos, especifique el directorio en el que residen los archivos jar
4. En la parte derecha de la página, seleccione los archivos jar de interés

## 5. Pulse en **Finalizar**

Los siguientes requisitos de archivo jar estarán en vigor:

- Un programa Java generado que accede a MQSeries de cualquier forma requiere clases MQ Series para Java; en concreto, el programa Java necesita los archivos jar siguientes (aunque no en el momento de la preparación):

- com.ibm.mq.jar
- com.ibm.mqbind.jar

Si tiene WebSphere MQ V5.2, el software está en IBM WebSphere MQ SupportPac MA88, que encontrará bajo MA88 en el sitio Web IBM (www.ibm.com). Baje e instale el software; entonces podrá acceder a los archivos jar desde el subdirectorio Java\lib del directorio en que haya instalado ese software.

Si tiene WebSphere MQ V5.3, puede obtener el software equivalente realizando una instalación personalizada y seleccionando Java Messaging. Entonces podrá acceder a los archivos jar desde el subdirectorio Java\lib del directorio de instalación de MQSeries.

- Un programa o envoltura Java generado que utilice el protocolo CICSJ2C para acceder a CICS para z/OS necesita acceder a connector.jar y cicsj2ee.jar, pero sólo durante la ejecución. Esos archivos estarán disponibles al instalar CICS Transaction Gateway.

**Nota:** El acceso a CICS es posible cuando el depurador Java EGL se ejecuta en J2EE. No obstante, se intentan las llamadas a CICS pero fallan cuando ese depurador se ejecuta fuera de J2EE o cuando está utilizando el depurador interpretativo de EGL, que siempre se ejecuta fuera de J2EE.

- Un programa Java generado que acceda a una tabla SQL requiere un archivo instalado con el sistema de gestión de base de datos.

- Para DB2 UDB, el archivo es uno de los siguientes:

```
sql1lib\java\db2java.zip
sql1lib\java\db2jcc.jar
```

El segundo de esos archivos está disponible con DB2 UDB Versión 8 o superior, como se describe en la documentación de DB2 UDB.

- Para Informix, los archivos son los siguientes:

```
ifxjdbc.jar
ifxjdbcx.jar
```

- Para Oracle, consulte la documentación de Oracle.

El archivo de base de datos es necesario durante la ejecución y puede utilizarse para validar sentencias de SQL en el momento de la preparación.

### **Tareas relacionadas**

“Establecer preferencias para el depurador de EGL” en la página 114

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354



---

## Consulta de EGL

---

### Compatibilidad de asignación en EGL

Las normas de compatibilidad de asignación (que se describen más adelante) se aplican en las siguientes situaciones:

- Cuando se asigna una variable que no es de referencia a otra; o bien
- Cuando EGL transfiere datos entre un argumento y el parámetro relacionado en una invocación de función, pero sólo si el parámetro de la función receptora tiene el modificador IN (en cuyo caso el argumento es el origen) o OUT (en cuyo caso el parámetro es el origen). Sin embargo, las normas de compatibilidad de asignación no se aplican si el parámetro está en la función `onPageLoad` de un `PageHandler`; para obtener detalles relativos a este caso, consulte el apartado *Compatibilidad de referencia en EGL*.

La compatibilidad de asignación se basa en la siguiente clasificación de tipos:

- Los tipos de texto son CHAR, MBCHAR, STRING y UNICODE
- Los tipos numéricos son BIN, INT, BIGINT, SMALLINT, DECIMAL, NUM, NUMBER, FLOAT, SMALLFLOAT, MONEY
- Los tipos de fecha y hora son DATE, INTERVAL, TIME, TIMESTAMP
- HEX tiene su propia categoría
- Los tipos de legado de VisualAge Generator son DBCHAR, NUMC y PACF, cada uno de los cuales sigue las normas de VisualAge Generator

Las normas de compatibilidad de asignación son las siguientes:

- Un campo de cualquier tipo de texto puede asignarse a un campo de cualquier tipo de texto
- Un campo de cualquier tipo numérico puede asignarse a un campo de cualquier tipo numérico
- Un campo de cualquier tipo de fecha y hora puede asignarse a un campo de cualquier tipo de texto o numérico
- Un campo de tipo STRING o CHAR puede asignarse a o desde un campo de tipo HEX
- Un campo de tipo CHAR puede asignarse a un campo de tipo NUM
- Para asignar un campo de tipo numérico a un campo de tipo texto, utilice la función de sistema **StrLib.formatNumber**
- Para asignar un campo de tipo DATE, TIME o TIMESTAMP a un campo formateado de un tipo de texto, utilice la función de sistema adecuada:
  - **StrLib.formatDate** (para fechas)
  - **StrLib.formatTime** (para horas)
  - **StrLib.formatTimestamp** (para indicaciones de la hora)
- Para asignar un campo de tipo texto a un campo de tipo DATE, TIME o TIMESTAMP, utilice la función de sistema adecuada:
  - **ConverseLib.dateValue** (para fechas)
  - **ConverseLib.timeValue** (para horas)
  - **ConverseLib.timestampValue** (para indicaciones de la hora)

## Asignación en diversos tipos numéricos

Un valor de cualquiera de los tipos numéricos (incluidos NUMC y PACF) puede asignarse a un campo de cualquier tipo numérico y tamaño, y EGL realizará las conversiones necesarias para conservar el valor en el formato destino.

Se añaden o truncan ceros no significativos si es necesario. (Los ceros iniciales de la parte entera de un valor no son significativos, al igual que los dígitos finales de la parte fraccionaria de un valor).

Para cualquiera de los tipos numéricos, puede utilizar la variable de sistema `sysVar.overflowIndicator` para comprobar si una asignación o un cálculo numérico han provocado un desbordamiento aritmético, y puede establecer la variable de sistema `VGVar.handleOverflow` para especificar la consecuencia de tal desbordamiento.

Si se produce un desbordamiento aritmético, el valor del campo destino no cambia. Si no se produce un desbordamiento aritmético, el valor asignado al campo destino se alinea de acuerdo con la declaración del campo destino.

Supongamos que está copiando un campo de tipo NUM en otro y que el valor de tiempo de ejecución del campo origen es 108.314:

- Si el campo destino permite siete dígitos con una posición decimal, el campo destino recibe el valor 000108.3, y *no* se detecta desbordamiento numérico. (Una pérdida de precisión en un valor fraccionario no se considera desbordamiento).
- Si el campo destino permite cuatro dígitos con dos posiciones decimales, se detecta un desbordamiento numérico y el valor del campo destino no cambia.

Cuando asigna un valor de coma flotante (tipo FLOAT o SMALLFLOAT) a un campo de un tipo de coma fija, el valor destino se trunca si es necesario. Si un valor origen es 108.357 y el destino de coma fija tiene un decimal, por ejemplo, el destino recibe 108.3.

## Otras asignaciones de tipos cruzados

A continuación se ofrecen detalles acerca de otras asignaciones de tipos cruzados:

- La asignación de un valor de tipo NUM a un destino de tipo CHAR sólo es válida si la declaración origen no tiene posiciones decimales. Esta operación es equivalente a una asignación de CHAR a CHAR.

Si la longitud origen es 4 y el valor es 21, por ejemplo, el contenido es equivalente a "0021", y una discrepancia de longitudes no provoca una condición de error:

- Si la longitud del destino es 5, el valor se almacena como "0021 " (se añade un espacio de un solo byte a la derecha)
- Si la longitud del destino es 3, el valor se almacena como "002 " (se trunca un dígito a la derecha)

Si el valor de tipo NUM es negativo y se asigna a un valor de tipo CHAR, el último byte copiado en el campo es un carácter no imprimible.

- La asignación de un valor de tipo CHAR a un destino de tipo NUM sólo es válida en el caso siguiente:
  - El origen (un campo o expresión de serie) contiene dígitos sin otros caracteres
  - La declaración destino no tiene posiciones decimales

Esta operación es equivalente a una asignación de NUM a NUM.

Si la longitud origen es 4 y el valor es "0021", por ejemplo, el contenido es equivalente a un 21 numérico; en los siguientes ejemplos se muestra el efecto de una discrepancia de longitudes:

- Si la longitud del destino es 5, el valor se almacena como 0021 (se añade un cero numérico a la izquierda)
  - Si la longitud del destino es 3, el valor se almacena como 021 (se trunca un dígito no significativo)
  - Si la longitud del destino es 1, el valor se almacena como 1
  - La asignación de un valor de tipo NUMC a un destino de tipo CHAR es posible en dos pasos, lo que elimina el signo si el valor es positivo:
    1. Asigne el valor NUMC a un destino de tipo NUM
    2. Asigne el valor NUM a un destino de tipo CHAR
- Si el valor del destino de tipo NUMC es negativo, el último byte copiado en destino de tipo CHAR es un carácter no imprimible.
- La asignación de un valor de tipo CHAR a un destino de tipo HEX sólo es válida si los caracteres del origen están dentro del rango de dígitos hexadecimales (0-9, A-F, a-f).
  - La asignación de un valor de tipo HEX a un destino de tipo CHAR almacena dígitos y letras mayúsculas (A-F) en el destino.
  - La asignación de un valor de tipo MONEY a un destino de tipo CHAR no es válida. El procedimiento recomendado para convertir desde MONEY a CHAR consiste en utilizar la función del sistema **strLib.formatNumber**.
  - La asignación de un valor de tipo NUM o CHAR a un destino de tipo DATE solo es válida si el valor origen es una fecha válida de acuerdo con la máscara *aaaaMMdd*; para obtener detalles, consulte el tema *DATE*.
  - La asignación de un valor de tipo NUM o CHAR a un destino de tipo TIME solo es válida si el valor fuente es una hora válida de acuerdo con la máscara *hhmmss*; para obtener detalles, consulte el tema *TIME*.
  - La asignación de un valor de tipo CHAR a un destino de tipo TIMESTAMP solo es válida si el valor origen es una indicación de la hora válida de acuerdo con la máscara del campo *TIMESTAMP*. A continuación se ofrece un ejemplo:

```
// NO válido porque el 30 de febrero no es una fecha válida
myTS timestamp("aaaaMMdd");
myTS = "20050230";
```

Si faltan los caracteres iniciales de una máscara completa (por ejemplo, si la máscara es "dd"), EGL supone que los caracteres de nivel superior ("aaaaMM", en este caso) representen el momento actual, de acuerdo con el reloj del sistema. Las sentencias siguientes originan un error de tiempo de ejecución en febrero:

```
// NO válido si se ejecuta en febrero
myTS timestamp("dd");
myTS = "30";
```

- La asignación de un valor de tipo TIME o DATE a un destino de tipo NUM es equivalente a una asignación NUM a NUM.
- La asignación de un valor de tipo TIME, DATE o TIMESTAMP a un destino de tipo CHAR es equivalente a una asignación CHAR a CHAR.

## Relleno y truncamiento en tipos de caracteres

Si el destino es de tipo carácter no *STRING* (incluidos *DBCHAR* y *HEX*) y tiene más espacio que el necesario para almacenar un valor origen, EGL lo rellena con datos por la derecha:

- Utiliza blancos de un solo byte para rellenar un destino de tipo *CHAR* o *MBCHAR*

- Utiliza blancos de doble byte para rellenar un destino de tipo DBCHAR
- Utiliza blancos de doble byte Unicode para rellenar un destino de tipo UNICODE
- Utiliza ceros binarios para rellenar un destino de tipo HEX, lo que significa (por ejemplo) que un valor origen "0A" se almacena en un destino de doble byte como "0A00" en lugar de "000A"

EGL trunca los valores por la derecha si el destino de un tipo de carácter no tiene espacio suficiente para almacenar el valor origen. No se indica ningún error. La siguiente situación representa un caso especial:

- La plataforma de entorno de ejecución da soporte al juego de caracteres EBCDIC
- La sentencia de asignación copia un literal de tipo MBCHAR o un elemento de tipo MBCHAR en un elemento más corto de tipo MBCHAR
- Un truncamiento byte por byte eliminaría un carácter de desplazamiento a teclado estándar final o dividiría un carácter DBCHAR

En esta situación, EGL trunca los caracteres según sea necesario para asegurarse de que el elemento destino contiene una serie válida de tipo MBCHAR y, a continuación, añade (si es necesario) blancos de un solo byte al final.

## Asignación entre indicaciones de la hora

Si asigna un elemento de tipo TIMESTAMP a otro campo de tipo TIMESTAMP, se aplican las normas siguientes:

- Si a la máscara del campo origen le faltan entradas de un nivel relativamente alto necesarias para el campo destino, las entradas de destino correspondientes se asignan de acuerdo con el reloj del sistema en el momento de la asignación, tal como se muestra en estos ejemplos:
 

```

- sourceTimeStamp timestamp ("MMdd");
 targetTimeStamp timestamp ("aaaaMMdd");

 sourceTimeStamp = "1201";

 // si este código se ejecuta en 2004, la sentencia siguiente
 // asigna 20041201 a targetTimeStamp
 targetTimeStamp = sourceTimeStamp;
- sourceTimeStamp02 timestamp ("ssff");
 targetTimeStamp02 timestamp ("mmssff");

 sourceTimeStamp02 = "3201";

 // la asignación siguiente incluye el minuto
 // en el que se ejecuta la sentencia de asignación
 targetTimeStamp02 = sourceTimeStamp02;
```
- Si a la máscara del campo origen le falta entradas de nivel relativamente bajo necesarias para el campo destino, a las entradas de destino correspondientes se les asignan los valores válidos más bajos, tal como muestran estos ejemplos:
 

```

- sourceTimeStamp timestamp ("aaaaMM");
 targetTimeStamp timestamp ("aaaaMMdd");

 sourceTimeStamp = "200412";

 // independientemente del día, la sentencia siguiente
 // asigna 20041201 a targetTimeStamp
 targetTimeStamp = sourceTimeStamp;
- sourceTimeStamp02 timestamp ("hh");
 targetTimeStamp02 timestamp ("hhmm");
```

```

sourceTimeStamp02 = "11";

// independientemente del minuto, la sentencia siguiente
// asigna 1100 a targetTimeStamp02
targetTimeStamp02 = sourceTimeStamp02;

```

## Asignación hacia o desde elementos subestructurados de estructuras fijas

Puede asignar un campo subestructurado a un campo no subestructurado o a la inversa, y puede asignar valores entre dos campos subestructurados. Supongamos, por ejemplo, que las variables denominadas *myNum* y *myRecord* se basan en los siguientes componentes:

```

DataItem myNumPart
 NUM(12)
end

Record myRecordPart type basicRecord
 10 topMost CHAR(4);
 20 next01 HEX(4);
 20 next02 HEX(4);
end

```

La asignación de un valor de tipo HEX a un elemento de tipo NUM no es válida fuera de las variables matemáticas del sistema; pero una asignación en el formato **myNum = topMost** es válida, debido a que **topMost** es de tipo CHAR. En términos generales, los tipos primitivos de los campos de una sentencia de asignación guían la asignación, y los tipos primitivos de los elementos subordinados no se tienen en cuenta.

El tipo primitivo de un elemento subestructurado es de tipo CHAR por omisión. Si asigna datos hacia o desde un campo subestructurado y no especifica un tipo primitivo diferente durante la declaración, las normas descritas anteriormente para los campos de tipo CHAR estarán en vigor durante la asignación.

## Asignación de un registro fijo

La asignación de un registro fijo a otro es equivalente a la asignación de un elemento subestructurado de tipo CHAR a otro. Una discrepancia de longitudes añade blancos de un solo byte a la derecha del valor recibido o elimina caracteres de un solo byte a la derecha del valor recibido. La asignación no tiene en cuenta los tipos primitivos de los campos de estructura subordinados.

Se aplican las siguientes excepciones:

- El contenido de un registro puede asignarse a un registro o a un campo de tipo CHAR, HEX o MBCHAR, pero no a un campo de ningún otro tipo
- Un registro puede recibir datos de un registro, de un literal de serie o de un campo de tipo CHAR, HEX o MBCHAR, pero no de un literal numérico ni de un campo de un tipo que no sea CHAR, HEX o MBCHAR

Finalmente, si asigna un registro SQL hacia o desde un registro de un tipo diferente, debe asegurarse de que el registro no SQL tenga espacio suficiente para el área de cuatro bytes que precede a cada campo de estructura.

### Conceptos relacionados

“PageHandler” en la página 194

### Consulta relacionada

"Asignaciones"

"Parámetros de función" en la página 522

"Componente de función en formato fuente EGL" en la página 527

"Componente PageHandler en formato fuente EGL" en la página 679

"Parámetros de programa" en la página 726

"Componente de programa en formato fuente EGL" en la página 728

### Consulta relacionada

"DATE" en la página 41

"Sentencias EGL" en la página 88

"formatNumber()" en la página 878

"handleOverflow" en la página 946

"move" en la página 610

"overflowIndicator" en la página 932

"Tipos primitivos" en la página 34

"Subseries" en la página 753

"TIME" en la página 43

---

## Asignaciones

Una asignación EGL copia datos de un área de memoria a otra y puede copiar el resultado de una expresión numérica o de texto en un campo origen.

► `origen = destino ;` ◄

### destino

Un campo, registro, registro fijo o variable de sistema

Puede especificar una subserie en el lado izquierdo de una sentencia assignment si el campo destino es de tipo CHAR, DBCHAR o UNICODE.

El área de subserie se rellena (con blancos, si es necesario) y el texto asignado no se extiende más allá del área de subserie, sino que se trunca si es necesario. Para obtener detalles acerca de la sintaxis, consulte *Subseries*.

**origen** Un registro, registro fijo o expresión numérica o de carácter

A continuación se ofrecen ejemplos de asignaciones:

```
z = a + b + c;
myDate = VVar.currentShortGregorianDate;
myUser = sysVar.userID;
myRecord01 = myRecord02;
myRecord02 = "USER";
```

El comportamiento de una sentencia assignment de EGL es distinto del de una sentencia **move** que se describe en el tema destinado a la sentencia *move*.

Las normas de la sentencia assignment se describen en el apartado *Compatibilidad de asignación en EGL*.

### Conceptos relacionados

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

### Consulta relacionada

"Compatibilidad de asignación en EGL" en la página 369

"move" en la página 610

"Subseries" en la página 753

---

## Elementos de asociación

Como se describen en las *Asociaciones de recursos*, el componente de asociaciones de recursos se compone de elementos de asociación. Cada elemento es específico de un nombre de archivo (propiedad "fileName" en la página 376) y contiene un conjunto de entradas, cada una de ellas con estas propiedades:

- "system" en la página 376
- "fileType"

Los valores de las propiedades **system** y **fileType** determinan las propiedades adicionales que están disponibles de las de la siguiente lista:

- "commit"
- "conversionTable"
- "formFeedOnClose" en la página 376
- "replace" en la página 376
- "systemName" en la página 377
- "text" en la página 377

### commit

Indica (para programas Java generados por EGL en iSeries) si debe habilitarse el control de compromiso.

Seleccione uno de los siguientes valores:

#### NO (valor por omisión)

La utilización de sysLib.commit o sysLib.rollback no tiene ningún efecto.

#### YES

Puede utilizar sysLib.commit y sysLib.rollback para definir el final de una unidad de trabajo lógica.

### conversionTable

Especifica el nombre de la tabla de conversión utilizada por un programa Java generado durante el acceso de una cola de mensajes MQSeries.

Para obtener más información, consulte el apartado *Conversión de datos*.

### fileType

Especifica la organización de archivos del sistema destino. Puede seleccionar un tipo explícito, como por ejemplo *seqws*. Como alternativa, puede seleccionar el valor *default*, que es el valor por omisión de la propiedad **fileType**. La utilización del valor por omisión significa que se seleccionará automáticamente un tipo de archivo:

- Para una combinación determinada de sistema destino y tipo de registro EGL; o
- Para salida impresa, cuando el nombre de archivo sea *printer*.

El apartado *Referencias cruzadas de tipos de registro y tipos de archivo* muestra los valores explícitos de **fileType**, así como el valor utiliza do si selecciona *default*.

## fileName

Hace referencia a un nombre de archivo lógico, según lo especificado en uno o varios registros. Está creando un elemento de asociación que relaciona este nombre con un recurso físico de uno o varios sistemas destino. (Para la salida impresa, especifique el valor *printer*.)

Puede utilizar un asterisco (\*) como carácter de sustitución global en un nombre de archivo lógico; sin embargo, ese carácter sólo es válido como último carácter. Para obtener detalles, consulte el apartado *Asociaciones de recursos y tipos de archivos*.

## formFeedOnClose

Indica si se emite una alimentación de papel cuando finaliza la salida de un formulario de impresión. (Un formulario de impresión se produce cuando el código emite una sentencia **print**).

Esta propiedad sólo está disponible si el valor de **fileName** es *printer* en uno de los siguientes casos:

- El valor de **system** es *aix*, *iSeriesj* o *linux*, y el valor de **fileType** es *seqws* o *spool*;  
o
- El valor de **system** es *win* y el valor de **fileType** es *seqws*.

Seleccione uno de los siguientes valores:

**YES**

Se produce una alimentación de papel (valor por omisión)

**NO**

No se produce una alimentación de papel

## replace

Especifica si, al añadir un registro al archivo, se sustituye el archivo en lugar de realizar adiciones al mismo. Esta entrada sólo se utiliza en estos casos:

- Está generando código Java; y
- El registro es del tipo de archivo **seqws**.

Seleccione uno de los siguientes valores:

**NO**

Se efectúan adiciones al archivo (valor por omisión)

**YES**

Se sustituye el archivo

## system

Especifica la plataforma destino. Seleccione uno de los siguientes valores:

**aix**

AIX

**iseriesj**

iSeries

**linux**

Linux

**win**

Windows 2000/NT/XP

**any**

Cualquier plataforma destino; para obtener detalles, consulte el apartado *Asociaciones de recursos y tipos de archivos*.

## **systemName**

Especifica el nombre de recurso del sistema del archivo o conjunto de datos asociado con el nombre de archivo. Especifique este valor entre comillas o apóstrofes si el valor incluye un espacio o alguno de los siguientes caracteres:

% = , ( ) /

## **text**

Especifica si debe provocarse que un programa Java generado haga lo siguiente al acceder a un archivo por medio de un registro serie:

- Añadir caracteres de fin de línea durante la operación **add**. En plataformas no UNIX, esos caracteres son los de retorno de carro y salto de línea; en plataformas UNIX, el único carácter es el salto de línea.
- Eliminar los caracteres de fin de línea durante las operaciones **get** o **get next**.

Seleccione uno de los siguientes valores:

**NO**

El valor por omisión es no añadir ni eliminar los caracteres de fin de línea

**YES**

Efectuar cambios, que resulta de utilidad si el programa generado intercambia datos con productos que esperan que los registros finalicen con los caracteres de fin de línea

### **Conceptos relacionados**

“Asociaciones de recursos y tipos de archivo” en la página 304

### **Tarea relacionada**

“Añadir componente asociaciones recursos a archivo de construcción EGL” en la página 309

“Editar componente de asociaciones recursos en archivo construcción EGL” en la página 309

“Eliminar componente asociaciones recursos de archivo construcción EGL” en la página 311

### **Consulta relacionada**

“Conversión de datos” en la página 467

“Valores de error de E/S” en la página 536

“Referencias cruzadas de tipo de registro y tipo de archivo” en la página 737

---

## **Elemento asynchLink**

Un elemento *asynchLink* de un componente de opciones de enlace especifica la forma en que un programa generado invoca otro programa de forma asíncrona, como ocurre cuando el programa originador invoca la función de sistema `sysLib.startTransaction`.

Puede evitar la necesidad de especificar un elemento *asynchLink* si acepta el comportamiento por omisión, que presupone que la transacción creada debe iniciarse desde el mismo paquete Java.

Cada elemento incluye la propiedad `recordName`, que hace referencia a un registro al que también se hace referencia en la función `sysLib.startTransaction` específica cuya acción se está modificando.

La otra propiedad es **package**, que sólo es necesaria si el código fuente del programa invocado se encuentra en un paquete que no es el del invocante.

#### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

#### Consulta relacionada

“Propiedad package del elemento asynchLink” en la página 379

“Propiedad recordName del elemento asynchLink” en la página 379

## Archivo **csoidpwd.properties** para llamadas remotas

En una situación que se describe más adelante, debe crear y suministrar acceso al archivo **csoidpwd.properties**. Ese archivo incluye detalles de autenticación necesarios para una llamada remota de un programa o envoltura Java.

La situación es la siguiente:

- El componente de opciones de enlace, elemento **callLink**, propiedad **remoteComType** está establecido en JAVA400, CICSJ2C o CICSECI; y
- Son necesarios un ID de usuario y una contraseña; y
- Se cumple una de estas condiciones:
  - La llamada se efectúa desde un programa Java, pero el código no invoca primero la función de sistema **SysLib.setRemoteUser** con valores que no sean blancos; o
  - La llamada se efectúa desde una envoltura Java, pero el código Java que incluye la envoltura no ha invocado los métodos de **CSOCallOptions** **setUserId** y **setPassword** con valores que no sean blancos.

Si la invocación de **SysLib.setRemoteUser** (o la invocación del método de **CSOCallOptions** adecuado) proporciona un ID de usuario o una contraseña en blanco, el valor de la propiedad equivalente se busca en **csoidpwd.properties**.

Su tarea es la siguiente:

1. Cree el archivo **csoidpwd.properties**, que puede contener valores de propiedad formateados del siguiente modo, cada uno en una línea independiente:  
**CSOUID=IDusuario**  
*IDusuario* es el ID de usuario de la llamada remota  
**CSOPWD=contraseña**  
*contraseña* es la contraseña de la llamada remota
2. Asegúrese de que el archivo está en un directorio referenciado en la vía de acceso de clases. Un directorio adecuado es la carpeta **JavaSource** del proyecto.

#### Conceptos relacionados

“Envoltura Java” en la página 301

#### Consulta relacionada

“Clases de envoltura Java” en la página 551

“Propiedad **remoteComType** del elemento **callLink**” en la página 419

“**setRemoteUser()**” en la página 908

## Propiedad package del elemento asynchLink

El componente de opciones de enlace, elemento asynchLink, propiedad **package** especifica el nombre del paquete que contiene el programa que se invoca. El valor por omisión es el paquete del programa invocante.

El nombre de paquete que se utiliza en los programas Java generados es el nombre de paquete del programa EGL, pero en minúsculas; y cuando EGL genera salida del elemento asynchLink, el valor de **package** se cambia (si es necesario) a minúsculas.

### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

### Consulta relacionada

“Elemento asynchLink” en la página 377

“Propiedad recordName del elemento asynchLink”

## Propiedad recordName del elemento asynchLink

El componente de opciones de enlace, elemento asynchLink, propiedad **recordName** especifica el nombre del registro utilizado en la función de sistema sysLib.startTransaction. En este caso, el nombre de registro se utiliza para identificar el programa o transacción que se asocia con el elemento asynchLink.

Puede utilizar un asterisco (\*) como carácter de sustitución global en el nombre de registro; sin embargo, ese carácter sólo es válido como último carácter. Para obtener detalles, consulte el apartado *Componente de opciones de enlace*.

### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

### Consulta relacionada

“Elemento asynchLink” en la página 377

“Propiedad package del elemento asynchLink”

“startTransaction()” en la página 911

---

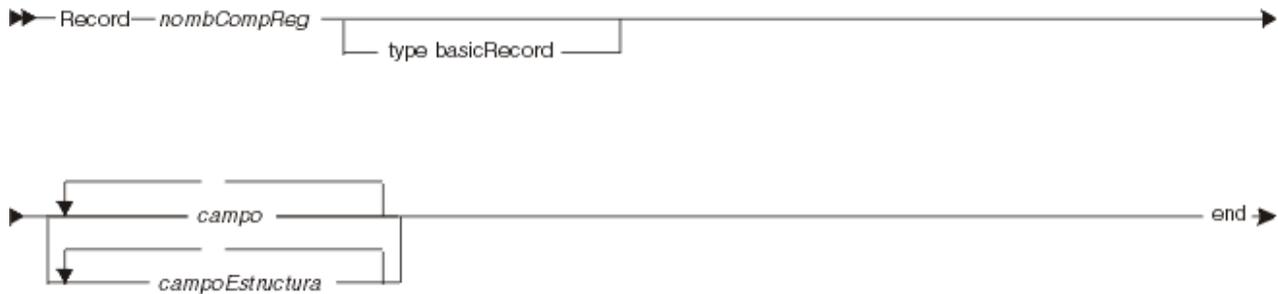
## Componente de registro básico en formato fuente EGL

Un componente de registro de tipo basicRecord se declara en un archivo EGL, como se describe en el apartado *Formato fuente EGL*.

A continuación se ofrece un ejemplo de componente de registro básico:

```
Record myBasicRecordPart type basicRecord
 10 myField01 CHAR(2);
 10 myField02 CHAR(78);
end
```

El diagrama de sintaxis de un componente de registro básico es el siguiente:



**Record nombreComponenteRegistro basicRecord**

Identifica el componente como de tipo basicRecord y especifica el nombre. Para conocer las normas, consulte el apartado *Convenios de denominación*.

*campo*

Una variable adecuada en un registro, tal como se describe en *Componentes de registro*. Finalizar cada declaración de variable con un punto y coma.

*campoEstructura*

Un campo de estructura fija, como se describe en la sección *Campo de estructura en formato fuente EGL*.

**Conceptos relacionados**

- “Proyectos, paquetes y archivos EGL” en la página 13
- “Componentes de registro fijo” en la página 133
- “Referencias a componentes” en la página 21
- “Componentes” en la página 17
- “Componentes de registro” en la página 132
- “Referencias a variables en EGL” en la página 58
- “Typedef” en la página 27

**Tareas relacionadas**

- “Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

**Consulta relacionada**

- “Componente DataItem en formato fuente EGL” en la página 472
- “Formato fuente EGL” en la página 491
- “Componente de función en formato fuente EGL” en la página 527
- “Componente de registro indexado en formato fuente EGL” en la página 535
- “Componente de registro MQ en formato fuente EGL” en la página 662
- “Convenios de denominación” en la página 672
- “Tipos primitivos” en la página 34
- “Componente de programa en formato fuente EGL” en la página 728
- “Propiedades que dan soporte a registros de longitud variable” en la página 737
- “Componente de registro relativo en formato fuente EGL” en la página 740
- “Componente de registro serie en formato fuente EGL” en la página 743
- “Componente de registro SQL en formato fuente EGL” en la página 748
- “Elemento de estructura en el formato fuente de EGL” en la página 752

---

## Componentes de construcción

### Formato de un archivo de construcción EGL

La estructura de un archivo .eglbl es el siguiente:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE EGL PUBLIC "-//IBM//DTD EGL 5.1//EN" "">
<EGL>
 <!-- colocar aquí las sentencias de importación -->
 <!-- colocar aquí los componentes -->
</EGL>

```

Su tarea consiste en colocar las sentencias de importación (import) y los componentes dentro del elemento <EGL>.

Debe especificar elementos <import> para hacer referencia al archivo que contiene el próximo descriptor de construcción de una serie o para hacer referencia a cualquiera de los componentes de construcción a los que hace referencia un descriptor de construcción. A continuación se ofrece un ejemplo de sentencia import:

```
<import file="myBldFile.egl.bld"/>
```

Debe declarar componentes de esta lista:

- <BuildDescriptor>
- <LinkageOptions>
- <ResourceAssociations>

A continuación se ofrece un ejemplo sencillo:

```

<EGL>
 <import file="myBldFile.egl.bld"/>
 <BuildDescriptor name="myBuildDescriptor"
 genProject="myNextProject"
 system="WIN"
 J2EE="NO"
 genProperties="GLOBAL"
 genDataTables="YES"
 dbms="DB2"
 sqlValidationConnectionURL="jdbc:db2:SAMPLE"
 sqlJDBCdriverClass="COM.ibm.db2.jdbc.app.DB2Driver"
 sqlDB="jdbc:db2:SAMPLE"
 </BuildDescriptor>
</EGL>

```

Puede revisar la DTD del archivo de construcción, que se encuentra en el siguiente subdirectorio:

```

dirInstalación\egl\eclipse\plugins\
com.ibm.etools.egl_versión\dtd

```

#### *DirInstalación*

El directorio de instalación del producto, como por ejemplo C:\Program Files\IBM\RSPD\6.0. Si instaló y tuvo un producto de Rational Developer antes de instalar el producto que está utilizando ahora, deberá especificar el directorio utilizado en la instalación anterior.

#### *versión*

La versión instalada del conector; por ejemplo, 6.0.0

El nombre del archivo (como por ejemplo egl\_wssd\_6\_0.dtd) empieza por las letras *egl* y un signo de subrayado. Los caracteres *wssd* hacen referencia a Rational Web Developer y Rational Application Developer, los caracteres *wsed* hacen referencia a Rational Application Developer para z/OS y los caracteres *wdsc* hacen referencia a Rational Application Developer para iSeries.

**Conceptos relacionados**

“Import” en la página 33

“Componentes” en la página 17

**Tareas relacionadas**

“Crear un archivo fuente EGL” en la página 128

**Consulta relacionada**

“Editor EGL” en la página 483

## Opciones del descriptor de construcción

La tabla siguiente lista todas las opciones del descriptor de construcción.

Opción del descriptor de construcción	Filtro(s) de opción de construcción	Descripción
<b>buildPlan</b>	<ul style="list-style-type: none"> <li>Destino Java</li> </ul>	Especifica si se crea un plan de construcción
<b>cicsj2cTimeout</b>	<ul style="list-style-type: none"> <li>Depuración</li> <li>Destino Java</li> <li>Java iSeries</li> </ul>	Asigna un valor a la propiedad de entorno de ejecución Java <b>cs0.cicsj2c.timeout</b> , que especifica el número de milisegundos que deben transcurrir antes de que se agote el tiempo de espera durante una llamada que utilice el protocolo CICSJ2C
<b>commentLevel</b>	<ul style="list-style-type: none"> <li>Destino Java</li> <li>Java iSeries</li> </ul>	Especifica el punto hasta el que se incluyen comentarios del sistema EGL en el código fuente de salida
<b>currencySymbol</b>	<ul style="list-style-type: none"> <li>Depuración</li> <li>Destino Java</li> </ul>	Especifica un símbolo de moneda que se compone de entre uno y tres caracteres
<b>dbms</b>	<ul style="list-style-type: none"> <li>Depuración</li> <li>Destino Java</li> <li>Java iSeries</li> </ul>	Especifica el tipo de base de datos a la que accede el programa generado
<b>decimalSymbol</b>	<ul style="list-style-type: none"> <li>Depuración</li> <li>Destino Java</li> <li>Java iSeries</li> </ul>	Asigna un carácter a la propiedad de entorno de ejecución Java <b>vgj.nls.number.decimal</b> , que indica el carácter que se utiliza como símbolo decimal
<b>destDirectory</b>	<ul style="list-style-type: none"> <li>Destino Java</li> </ul>	Especifica el nombre del directorio que almacena la salida de preparación, pero sólo al generar Java
<b>destHost</b>	<ul style="list-style-type: none"> <li>Destino Java</li> </ul>	Especifica el nombre o la dirección numérica TCP/IP de la máquina destino en la que reside el servidor de construcción

Opción del descriptor de construcción	Filtro(s) de opción de construcción	Descripción
<b>destPassword</b>	<ul style="list-style-type: none"> <li>• Destino Java</li> </ul>	Especifica la contraseña utilizada por EGL para iniciar la sesión en la máquina en la que se produce la preparación
<b>destPort</b>	<ul style="list-style-type: none"> <li>• Destino Java</li> </ul>	Especifica el puerto en el que un servidor de construcción remota está a la escucha de las peticiones de construcción
<b>destUserID</b>	<ul style="list-style-type: none"> <li>• Destino Java</li> </ul>	Especifica el ID de usuario utilizado por EGL para iniciar la sesión en la máquina en la que se produce la preparación
<b>eliminateSystemDependentCode</b>	<ul style="list-style-type: none"> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Indica si, durante la validación, EGL pasa por alto el código que nunca se ejecutará en el sistema destino
<b>enableJavaWrapperGen</b>	<ul style="list-style-type: none"> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Especifica si se permite la generación de clases de envoltura Java
<b>genDataTables</b>	<ul style="list-style-type: none"> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Indica si se desea generar tablas de datos
<b>genDirectory</b>	<ul style="list-style-type: none"> <li>• Destino Java</li> </ul>	Especifica la vía de acceso totalmente calificada la directorio en el que EGL coloca la salida generada y los archivos de estado de la preparación
<b>genFormGroup</b>	<ul style="list-style-type: none"> <li>• Destino Java</li> </ul>	Indica si desea generar el grupo de formularios al que se hace referencia en la declaración de uso del programa que está generando
<b>genHelpFormGroup</b>	<ul style="list-style-type: none"> <li>• Destino Java</li> </ul>	Indica si desea generar el grupo de formularios de ayuda al que se hace referencia en la declaración de uso del programa que está generando
<b>genProject</b>	<ul style="list-style-type: none"> <li>• Destino Java</li> </ul>	Coloca la salida de la generación Java en un proyecto del entorno de trabajo y automatiza las tareas necesarias para la configuración del entorno de ejecución Java

Opción del descriptor de construcción	Filtro(s) de opción de construcción	Descripción
<b>genProperties</b>	<ul style="list-style-type: none"> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Especifica el tipo de propiedades de entorno de ejecución Java (si las hay) y, en algunos casos, si debe generarse un archivo de propiedades de enlace
<b>itemsNullable</b>	<ul style="list-style-type: none"> <li>• Depuración</li> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Especifica la circunstancia en la que el código puede establecer campos primitivos en NULL
<b>J2EE</b>	<ul style="list-style-type: none"> <li>• Depuración</li> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Especifica si un programa Java se genera para ejecutarse en un entorno J2EE
<b>linkage</b>	<ul style="list-style-type: none"> <li>• Depuración</li> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Contiene el nombre del componente de opciones de enlace que controla los aspectos de la generación
<b>nextBuildDescriptor</b> (consulte el apartado Componente descriptor de construcción)	<ul style="list-style-type: none"> <li>• Depuración</li> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Identifica el siguiente descriptor de construcción de la serie
<b>prep</b>	<ul style="list-style-type: none"> <li>• Destino Java</li> </ul>	Especifica si EGL inicia la preparación cuando la generación finaliza con un código de retorno $\leq 4$
<b>resourceAssociations</b>	<ul style="list-style-type: none"> <li>• Depuración</li> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Contiene el nombre de un componente de asociaciones de recurso, que relaciona componentes de registro con archivos y colas de las plataformas destino
<b>sessionBeanID</b>	<ul style="list-style-type: none"> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Identifica el nombre de un elemento de sesión existente en el descriptor de despliegue J2EE
<b>sqlCommitControl</b>	<ul style="list-style-type: none"> <li>• Depuración</li> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Permite la generación de una propiedad de tiempo de ejecución Java que especifica si se produce un compromiso después de cada cambio en la base de datos por omisión
<b>sqlDB</b>	<ul style="list-style-type: none"> <li>• Depuración</li> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Especifica la base de datos por omisión utilizada por un programa generado
<b>sqlID</b>	<ul style="list-style-type: none"> <li>• Depuración</li> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Especifica un ID de usuario utilizado para conectarse a una base de datos durante la validación en tiempo de generación de las sentencias SQL o durante la ejecución

Opción del descriptor de construcción	Filtro(s) de opción de construcción	Descripción
<b>sqlJDBCClass</b>	<ul style="list-style-type: none"> <li>• Depuración</li> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Especifica una clase de controlador utilizada para conectarse a una base de datos durante la validación en tiempo de generación de las sentencias SQL o durante una sesión de depuración Java no J2EE
<b>sqlJNDIName</b>	<ul style="list-style-type: none"> <li>• Depuración</li> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Especifica la base de datos por omisión utilizada por un programa Java generado que se ejecuta en J2EE
<b>sqlPassword</b>	<ul style="list-style-type: none"> <li>• Depuración</li> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Especifica una contraseña utilizada para conectarse a una base de datos durante la validación en tiempo de generación de las sentencias SQL o durante la ejecución
<b>sqlValidationConnectionURL</b>	<ul style="list-style-type: none"> <li>• Depuración</li> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Especifica un URL utilizado para conectarse a una base de datos durante la validación en tiempo de generación de las sentencias SQL
<b>system</b>	<ul style="list-style-type: none"> <li>• Depuración</li> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Especifica una categoría de salida de generación
<b>targetNLS</b>	<ul style="list-style-type: none"> <li>• Depuración</li> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Especifica el código de idioma nacional destino utilizado para la salida de tiempo de ejecución
<b>VAGCompatibility</b>	<ul style="list-style-type: none"> <li>• Depuración</li> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Indica si el proceso de generación permite la utilización de sintaxis de programa especial
<b>validateSQLStatements</b>	<ul style="list-style-type: none"> <li>• Destino Java</li> <li>• Java iSeries</li> </ul>	Indica si las sentencias SQL se validan con una base de datos

#### Conceptos relacionados

“Componente descriptor de construcción” en la página 293

“Propiedades de tiempo de ejecución Java” en la página 347

#### Tareas relacionadas

“Añadir componente descriptor construcción a archivo construcción EGL” en la página 298

“Editar opciones generales en un descriptor de construcción” en la página 299

#### Consulta relacionada

“Propiedades de ejecución de Java (detalles)” en la página 540

#### buildPlan

La opción **buildPlan** del descriptor de construcción especifica si se crea un plan de construcción. Los valores válidos son YES y NO, y el valor por omisión es YES.

El plan de construcción se coloca en el directorio identificado por la opción **genDirectory** del descriptor de construcción.

Un caso especial se produce cuando se genera código Java en un proyecto. En este caso no se crea ningún plan de construcción independientemente del valor de **buildPlan**, pero la preparación tiene lugar en cualquiera de estas dos situaciones:

- Siempre que se reconstruye el proyecto
- Siempre que se generan los archivos fuente; pero sólo si ha marcado la preferencia del entorno de trabajo **Realizar construcción automáticamente al modificar el recurso**

Puede que desee crear un plan de construcción e invocar ese plan más tarde. Para obtener detalles, consulte el apartado *Invocar un plan de construcción después de la generación*.

#### **Conceptos relacionados**

“Plan de construcción” en la página 327

#### **Tareas relacionadas**

“Invocar un plan de construcción tras la generación” en la página 334

#### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

#### **cicsj2cTimeout**

Cuando se genera código Java, la opción **cicsj2cTimeout** del descriptor de construcción asigna un valor a la propiedad de entorno de ejecución Java **cso.cicsj2c.timeout**. Esa propiedad especifica el número de milisegundos que deben transcurrir antes de que se agote el tiempo de espera durante una llamada que utilice el protocolo CICSJ2C.

El valor por omisión de la propiedad de entorno de ejecución es 30000, que representa 30 segundos. Si el valor se establece en 0, el tiempo de espera no se agota. El valor debe ser superior o igual a 0.

La propiedad **cso.cicsj2c.timeout** no afecta a las llamadas si el programa llamado se ejecuta en WebSphere 390; para obtener detalles, consulte el apartado *Configurar el servidor J2EE para llamadas CICSJ2C*.

#### **Conceptos relacionados**

“Propiedades de tiempo de ejecución Java” en la página 347

#### **Tareas relacionadas**

“Configuración del servidor J2EE para llamadas CICSJ2C” en la página 358

#### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

“Propiedades de ejecución de Java (detalles)” en la página 540

#### **commentLevel**

La opción **commentLevel** del descriptor de construcción especifica el punto hasta el que se incluyen comentarios del sistema EGL en el código fuente de salida.

Los valores válidos son los siguientes:

- 0 La salida incluye comentarios mínimos, los que incluye comentarios acerca de los alias de nombre generados por EGL
- 1 Además de los comentarios incluidos en el nivel 0, se colocan sentencias de script inmediatamente antes del código generado para implementar dichas sentencias.

El valor por omisión es 1.

El hecho de aumentar el nivel de comentarios afecta de ningún modo al tamaño o al rendimiento del código preparado, pero aumenta el tamaño de la salida y el tiempo necesario para generarla, transferirla y prepararla.

#### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

### **currencySymbol**

La opción **currencySymbol** del descriptor de construcción sólo está disponible para la salidaJava y especifica un símbolo de moneda que se compone de entre uno y tres caracteres. Si no especifica esta opción, el valor por omisión se deriva del entorno local del sistema en el que se genera la salida.

Para especificar un carácter que no se encuentre en el teclado, mantenga pulsada la tecla **Alt** y utilice el teclado numérico para escribir el código decimal del carácter. El código decimal para el Euro, por ejemplo, es 0128 en Windows 2000/NT/XP.

#### **Conceptos relacionados**

“Componente descriptor de construcción” en la página 293

#### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

### **dbms**

La opción **dbms** del descriptor de construcción especifica el tipo de base de datos a la que accede el programa generado. Seleccione uno de los siguientes valores:

- DB2 (valor por omisión)
- INFORMIX
- ORACLE

#### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

“Informix y EGL” en la página 252

### **decimalSymbol**

Cuando se genera código Java, la opción **decimalSymbol** del descriptor de construcción asigna un carácter a la propiedad de entorno de ejecución Java **vgj.number.decimal**. Si no especifica la opción **decimalSymbol** del descriptor de construcción, el carácter queda determinado por el entorno local asociado con la propiedad de entorno de ejecución Java **vgj.nls.code**.

El valor no puede tener más de un carácter.

#### **Conceptos relacionados**

“Propiedades de tiempo de ejecución Java” en la página 347

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“Propiedades de ejecución de Java (detalles)” en la página 540

### destDirectory

La opción **destDirectory** del descriptor de construcción especifica el directorio que almacena la salida de preparación. Esta opción sólo es relevante al generar en un directorio en lugar de en un proyecto.

Cuando especifique una vía de acceso de archivo totalmente calificada, debe existir toda excepto el último directorio. Si especifica `c:\buildout` en Windows 2000, por ejemplo, EGL crea el directorio `buildout`, si no existe. Sin embargo, si especifica `c:\interim\buildout` y el directorio `interim` no existe, la preparación fallará.

Si especifica un directorio relativo (como por ejemplo `myid/mysource` en USS), la salida se coloca en el directorio situado más abajo, que es relativo al directorio por omisión, como se describe a continuación.

El valor por omisión de **destDirectory** resulta afectado por el estado de la opción **destHost** del descriptor de construcción:

- Si se especifica **destHost**, el valor por omisión de **destDirectory** es el directorio en el que se ha iniciado el servidor de construcción
- Si no se especifica **destHost**, la preparación tiene lugar en la máquina en la que se produce la generación, y el valor por omisión de **destDirectory** los suministra la opción **genDirectory** del descriptor de construcción

El usuario especificado por la opción **destUserID** del descriptor de construcción debe tener autorización de escritura sobre el directorio que recibe la salida de la preparación.

No puede utilizar una variable UNIX (`$HOME`, por ejemplo) para identificar parte de una estructura de directorios en USS.

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“**destHost**”

“**genProject**” en la página 393

### destHost

La opción **destHost** del descriptor de construcción especifica el nombre o la dirección numérica TCP/IP de la máquina destino en la que reside el servidor de construcción. No hay ningún valor por omisión disponible.

Si esta preparando salida Java, se aplican las siguientes normas:

- **destHost** es opcional
- **destHost** sólo es relevante al generar en un directorio en lugar de en un proyecto
- Si especifica **destHost** sin especificar **destDirectory**, el directorio en el que se ha iniciado el servidor de construcción es el que recibe las salidas origen y de preparación
- Si no especifica **destHost**, la preparación tiene lugar en la máquina en la que se produce la generación; y si no especifica **destDirectory**, el directorio especificado por la opción **genDirectory** del descriptor de construcción es el que recibe las salidas origen y de preparación

- Los entornos UNIX son sensibles a mayúsculas y minúsculas

Puede escribir 64 caracteres como máximo para el nombre o la dirección TCP/IP. Si está desarrollando en Windows NT, debe especificar un nombre en lugar de una dirección TCP/IP.

A continuación figuran dos ejemplos de valor de **destHost**:

```
abc.def.ghi.com
```

```
9.99.999.99
```

#### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

“destDirectory” en la página 388

“destPassword”

“destPort”

#### **destPassword**

La opción **destPassword** del descriptor de construcción especifica la contraseña utilizada por EGL para iniciar la sesión en la máquina en la que se produce la preparación.

Esta opción y la descripción de esta página sólo son relevantes si está generando en un directorio en lugar de en un proyecto y sólo si especifica un valor para la opción **destHost** del descriptor de construcción.

La contraseña proporciona acceso al ID de usuario especificado en la opción **destUserID** del descriptor de construcción. El valor de la contraseña es sensible a mayúsculas y minúsculas en todas las plataformas destino.

No hay ningún valor por omisión disponible.

La utilización de **destPassword** significa que se almacenará una contraseña en un archivo de construcción EGL. Puede evitar el riesgo de seguridad no estableciendo la opción del descriptor de construcción. Al iniciar la generación, puede establecer la contraseña en un diálogo de generación interactivo o en la línea de mandatos.

#### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

“destHost” en la página 388

“destUserID” en la página 390

#### **destPort**

La opción **destPort** del descriptor de construcción especifica el puerto en el que un servidor de construcción remota está a la escucha de las peticiones de construcción.

Esta opción sólo es relevante si está generando en un directorio en lugar de en un proyecto y sólo si especifica un valor para la opción **destHost** del descriptor de construcción.

No hay ningún valor por omisión disponible.

#### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

“destHost” en la página 388

## destUserID

La opción **destUserID** del descriptor de construcción especifica el ID de usuario utilizado por EGL para iniciar la sesión en la máquina en la que se produce la preparación.

Esta opción y la descripción de esta página sólo son relevantes si está generando en un directorio en lugar de en un proyecto y sólo si especifica un valor para la opción **destHost** del descriptor de construcción.

El usuario especificado por la opción **destUserID** debe tener autorización de escritura sobre el directorio. El valor de esta opción es sensible a mayúsculas y minúsculas en todos los sistemas destino.

No hay ningún valor por omisión disponible.

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“destHost” en la página 388

“destPassword” en la página 389

## eliminateSystemDependentCode

La opción **eliminateSystemDependentCode** del descriptor de construcción indica si, durante la validación, EGL pasa por alto el código que nunca se ejecutará en el sistema destino. Los valores válidos son *yes* (el valor por omisión) y *no*. Especifique *no* solo si la salida de la generación actual se va a ejecutar en varios sistemas.

La opción **eliminateSystemDependentCode** sólo es relevante en relación a la función de sistema **sysVar.systemType**. Dicha función no afecta por sí misma al código que se valida durante la generación. Por ejemplo, la siguiente sentencia **add** pueda validarse aunque esté generando para Windows:

```
if (sysVar.systemType IS AIX)
 add myRecord;
end
```

Para evitar validar el código que nunca se ejecutará en el sistema destino, realice cualquiera de las siguientes acciones:

- Establezca la opción **eliminateSystemDependentCode** del descriptor de construcción en *yes*. En el ejemplo actual, la sentencia **add** no se valida si establece la opción del descriptor de construcción en *yes*. Sin embargo, tenga en cuenta que el generador sólo puede eliminar el código que depende del sistema si la expresión lógica (en este caso, `sysVar.systemType IS AIX`) es la suficientemente simple para evaluarse durante la generación.
- Como alternativa, traslade las sentencias que no desee validar a un segundo programa; a continuación, deje que el programa original llame al programa nuevo de forma condicional:

```
if (sysVar.systemType IS AIX)
 call myAddProgram myRecord;
end
```

### Conceptos relacionados

“Componente descriptor de construcción” en la página 293

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

## **enableJavaWrapperGen**

Cuando se emiten los mandatos para generar un programa, la opción **enableJavaWrapperGen** del descriptor de construcción permite elegir entre tres alternativas:

### **YES (valor por omisión)**

Generar el programa y permitir la generación de las clases de envoltura Java relacionadas y (si procede) del bean de sesión EJB relacionado

### **ONLY**

No generar el programa, pero permitir la generación de las clases de envoltura Java relacionadas y (si procede) del bean de sesión EJB relacionado

### **NO**

Generar el programa, pero no las clases de envoltura Java ni el bean de sesión EJB relacionado, si existe

La generación real de las clases de envoltura Java y del bean de sesión EJB requiere especificar los valores adecuados en el componente de opciones de enlace utilizado durante la generación. Para obtener una visión general, consulte el apartado *Envoltura Java*.

### **Conceptos relacionados**

“Envoltura Java” en la página 301

### **Consulta relacionada**

“Clases de envoltura Java” en la página 551

## **genDataTables**

La opción **genDataTables** del descriptor de construcción indica si desea generar las tablas de datos a las que se hace referencia en el programa que está generando. Las referencias se encuentran en la declaración de uso del programa y en la propiedad de programa **msgTablePrefix**.

Los valores válidos son *yes* (el valor por omisión) y *no*.

Establezca el valor en *no* en el caso siguiente:

- Las tablas de datos referenciadas en el programa se han generado anteriormente; y
- Dichas tablas no han cambiado desde que se generaron por última vez.

Para obtener otros detalles, consulte el apartado *Componente DataTable*.

### **Conceptos relacionados**

“Componente descriptor de construcción” en la página 293

“DataTable” en la página 146

### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

“Componente de programa en formato fuente EGL” en la página 728

“Declaración use” en la página 954

## **genDirectory**

La opción **genDirectory** del descriptor de construcción especifica la vía de acceso totalmente calificada la directorio en el que EGL coloca la salida generada y los archivos de estado de la preparación.

Al generar en el entorno de trabajo o desde la interfaz de proceso por lotes del entorno de trabajo, se aplican las siguientes normas:

#### **Para la generación Java**

Debe especificar **genProject** o **genDirectory**, pero se producirá un error si especifica ambas. También debe especificar **genProject** si genera código para iSeries.

Si genera desde el SDK de EGL, se aplican las siguientes normas:

- Debe especificar **genDirectory**
- Si especifica **genProject**, se produce un error
- No puede generar código Java para iSeries

Para obtener detalles acerca del despliegue de código Java, consulte la sección *Procesar código Java generado en un directorio*.

#### **Conceptos relacionados**

“Generación a partir del SDK (Software Development Kit) de EGL” en la página 333

“Generación a partir de la interfaz por lotes del entorno de trabajo” en la página 332

“Generación en el entorno de trabajo” en la página 330

#### **Tareas relacionadas**

“Procesar código Java generado en un directorio” en la página 335

#### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

“genDirectory” en la página 391

“genProject” en la página 393

### **genFormGroup**

La opción **genFormGroup** del descriptor de construcción indica si desea generar el grupo de formularios a los que se hace referencia en la declaración de uso del programa que está generando. Los valores válidos son *yes* (el valor por omisión) y *no*.

El grupo de formularios de ayuda, si existe, no resulta afectado por esta opción, sino por la opción **genHelpFormGroup** del descriptor de construcción.

#### **Conceptos relacionados**

“Componente descriptor de construcción” en la página 293

#### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

“genHelpFormGroup”

“Declaración use” en la página 954

### **genHelpFormGroup**

La opción **genHelpFormGroup** del descriptor de construcción indica si desea generar el grupo de formularios de ayuda a los que se hace referencia en la declaración de uso del programa que está generando. Los valores válidos son *yes* (el valor por omisión) y *no*.

El grupo de formularios principal no resulta afectado por esta opción, sino por la opción **genFormGroup** del descriptor de construcción.

## Conceptos relacionados

“Componente descriptor de construcción” en la página 293

## Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“genFormGroup” en la página 392

“Declaración use” en la página 954

## genProject

La opción **genProject** del descriptor de construcción coloca la salida de la generación Java en un proyecto del entorno de trabajo y automatiza las tareas necesarias para la configuración del entorno de ejecución Java. Para obtener detalles acerca de dicha configuración y las ventajas de utilizar **genProject**, consulte el apartado *Generación de código Java en un proyecto*.

Para utilizar **genProject**, especifique el nombre de proyecto. A continuación, EGL pasará por alto las opciones **buildPlan**, **genDirectory** y **prep** del descriptor de construcción, y la preparación se producirá en cualquiera de estos dos casos:

- Siempre que se reconstruye el proyecto
- Siempre que se generan los archivos fuente; pero sólo si ha marcado la preferencia del entorno de trabajo **Realizar construcción automáticamente al modificar el recurso**

Si establece la opción **genProject** en el nombre de un proyecto que no existe en el entorno de trabajo, EGL utiliza el nombre para crear un proyecto Java, excepto en los siguientes casos:

- Si está generando un PageHandler y especifica un proyecto diferente del que contiene el JSP relacionado y si ese otro proyecto no existe, EGL crea un proyecto Web EGL. (Sin embargo, es aconsejable generar el PageHandler en el proyecto que contiene el JSP relacionado).
- Una segunda excepción concierne al proceso de EJB y se produce si está generando una envoltura Java cuando el componente de opciones de enlace, elemento callLink, propiedad **type** es **ejbCall** (para la llamada desde la envoltura al programa generado por EGL). En ese caso, EGL utiliza el valor de **genProject** para crear un proyecto EJB y crea un proyecto de aplicación de empresa nuevo (si es necesario) con el mismo nombre que el del proyecto EJB más las letras EAR.

Además de crear un proyecto, EGL hace lo siguiente:

- EGL crea carpetas en el proyecto. La estructura de paquetes empieza bajo la carpeta de nivel superior JavaSource. Puede cambiar el nombre JavaSource pulsando con el botón derecho del ratón en el nombre de carpeta y seleccionando **Propagar**.
- Si se especifica una definición JRE en la página de preferencias de Java (JRE instalados), EGL añade la variable de vía de acceso de clases JRE\_LIB. Esa variable contiene la vía de acceso a los archivos JAR de tiempo de ejecución del JRE utilizado actualmente.

Al generar en el entorno de trabajo o desde la interfaz de proceso por lotes del entorno de trabajo, se aplican las siguientes normas:

### Para la generación Java

No es necesario que especifique **genProject** o **genDirectory**. Si no especifica ninguna de las dos, la salida Java se genera en el proyecto que contiene el archivo fuente EGL que se genera.

Si está generando un PageHandler y el proyecto especificado existe, el proyecto debe ser de tipo Web EGL. Si está generando un bean de sesión EJB y el proyecto especificado existe, el proyecto debe ser de tipo EJB.

#### Para generación COBOL

Debe especificar **genDirectory** y, EGL pasará por alto cualquier valor para **genProject**.

Si genera desde el SDK de EGL, se aplican las siguientes normas:

- Debe especificar **genDirectory**
- Si especifica **genProject**, se produce un error
- No puede generar código Java para iSeries

#### Conceptos relacionados

“Generación a partir del SDK (Software Development Kit) de EGL” en la página 333

“Generación a partir de la interfaz por lotes del entorno de trabajo” en la página 332

“Generación en el entorno de trabajo” en la página 330

“Generación de código Java en un proyecto” en la página 323

#### Tareas relacionadas

##### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“buildPlan” en la página 385

“genDirectory” en la página 391

“prep” en la página 397

“Propiedad type del elemento callLink” en la página 423

#### genProperties

La opción **genProperties** del descriptor de construcción especifica el tipo de propiedades de entorno de ejecución Java (si las hay) y, en algunos casos, si debe generarse un archivo de propiedades de enlace. Esta opción del descriptor de construcción sólo es relevante al generar un programa Java (que puede utilizar cualquier tipo de salida) o una envoltura (que sólo puede utilizar el archivo de propiedades de enlace

Los valores válidos son los siguientes:

#### NO (valor por omisión)

EGL no genera propiedades de entorno de ejecución ni de enlace.

#### PROGRAM

Los resultados son los siguientes:

- Si está generando un programa para que se ejecute fuera de J2EE, EGL genera un archivo de propiedades que es específico del programa que se genera. El nombre de ese archivo es el siguiente:

```
pgmAlias.properties
```

```
pgmAlias
```

El nombre del programa durante la ejecución.

- Los demás resultados se producen si especifica **PROGRAM** o **GLOBAL**:
  - Si está generando un programa que se ejecuta en J2EE, EGL genera un archivo de entorno J2EE o en un descriptor de despliegue; para obtener detalles, consulte el apartado *Alternativas para establecer valores del descriptor de despliegue*.

- Si genera una envoltura o un programa llamador Java, EGL puede generar un archivo de propiedades de enlace; para obtener detalles acerca de la situación en la que se genera este archivo, consulte el apartado *Archivo de propiedades de enlace (referencia)*.

## GLOBAL

Los resultados son los siguientes:

- Si está generando un programa para que se ejecute fuera de J2EE, EGL genera un archivo de propiedades que se utiliza en toda la unidad de ejecución, pero que no se indica para el programa inicial de la unidad de ejecución. El nombre de ese archivo de propiedades es **rununit.properties**.

Esta opción resulta especialmente útil cuando el primer programa de la unidad de ejecución no accede a un archivo o base de datos, pero llama a programas que sí lo hacen.

Al generar el llamador, puede generar un archivo de propiedades indicado para el programa, y el contenido puede no incluir propiedades relacionadas con la base de datos. Al generar el programa llamado, puede generar el archivo **rununit.properties**, y el contenido estará disponible para ambos programas.

- Los demás resultados se producen si especifica **GLOBAL** o **PROGRAM**:
  - Si está generando un programa que se ejecuta en J2EE, EGL genera un archivo de entorno J2EE o en un descriptor de despliegue; para obtener detalles, consulte el apartado *Alternativas para establecer valores del descriptor de despliegue*.
  - Si genera una envoltura o un programa llamador Java, EGL puede generar un archivo de propiedades de enlace; para obtener detalles acerca de la situación en la que se genera este archivo, consulte el apartado *Archivo de propiedades de enlace (referencia)*.

Para obtener más detalles, consulte los apartados *Propiedades de entorno de ejecución Java* y *Archivo de propiedades de enlace*.

### Conceptos relacionados

“Archivo de entorno J2EE” en la página 357

“Propiedades de tiempo de ejecución Java” en la página 347

“Componente de opciones de enlace” en la página 311

“Archivo de propiedades de enlace” en la página 364

### Tareas relacionadas

“Opciones del descriptor de construcción” en la página 382

“Establecer valores de descriptor de despliegue” en la página 355

### Consulta relacionada

“Propiedades de ejecución de Java (detalles)” en la página 540

## itemsNullable

La opción de descriptor de construcción **itemsNullable** especifica la circunstancia en la que el código puede establecer campos primitivos en NULL.

Los valores válidos son los siguientes:

### NO

No puede establecer campos primitivos en NULL excepto en este caso:

- El campo es un registro SQL y
- La propiedad del elemento SQL **isNullable** se establece en yes.

Este valor de **itemsNullable** es el valor predeterminado y el comportamiento es coherente con las versiones anteriores de EGL.

### YES

Puede establecer en NULL cualquier campo primitivo de cualquier registro que no sea un registro fijo. El comportamiento es coherente con el producto I4GL de Informix.

La tabla siguiente muestra el efecto de la decisión.

Tabla 9. Efecto de **itemsNullable**

Operación	ItemsNullable = NO	ItemsNullable = YES
Asignar un campo nulo a otro campo	El valor del origen es 0 o espacio en blanco y la asignación copia tanto un valor (si el destino tiene capacidad de nulo) como el estado NULL.	Si el destino tiene capacidad de nulo, se establece en NULL. De lo contrario, el destino se establece en 0 o en un espacio en blanco.
Utilizar un campo nulo en una expresión numérica	El campo se trata como si contuviera un 0	La expresión se resuelve en NULL
Utilizar un campo nulo en una expresión de texto	El campo se trata como si contuviera un espacio	El campo se trata como si hubiera una serie vacía
Utilizar un campo nulo en una expresión lógica	La expresión se trata como si el valor del campo fuera 0 o un espacio en blanco, con el ejemplo siguiente que se resuelve en TRUE:  0 == null	La expresión se resuelve en TRUE solo si nulo se compara con nulo, como no es el caso en el ejemplo siguiente, que se resuelve en FALSE:  0 == null
Establecer campo vacío	El estado nulo es no establecido	El estado nulo es establecido
Establecer registro vacío	El estado nulo es no establecido	El estado nulo es establecido

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

### J2EE

La opción **J2EE** del descriptor de construcción especifica si un programa Java se genera para ejecutarse en un entorno J2EE. Los valores válidos son los siguientes:

#### NO (valor por omisión)

Genera un programa que no se ejecutará en un entorno J2EE. El programa se conecta a las bases de datos directamente y el entorno está definido por un archivo de propiedades.

#### YES

Genera un programa que se ejecutará en un entorno J2EE. El programa se conecta a las bases de datos mediante un origen de datos y el entorno está definido por un descriptor de despliegue.

Cuando se genera un PageHandler, J2EE está siempre establecido en YES independientemente de lo especificado en esta opción.

### Conceptos relacionados

“Depurador EGL” en la página 279

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

### linkage

La opción **linkage** del descriptor de construcción contiene el nombre del componente de opciones de enlace que controla los aspectos de la generación. Esta opción no es obligatoria para la generación y no hay ningún valor por omisión disponible.

### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“Elemento callLink” en la página 407

### nextBuildDescriptor

La opción **nextBuildDescriptor** del descriptor de construcción identifica el siguiente descriptor de construcción de la serie, si existe. Para obtener detalles, consulte el apartado *Componente descriptor de construcción*.

### Conceptos relacionados

“Componente descriptor de construcción” en la página 293

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

### prep

La opción **prep** del descriptor de construcción especifica si EGL inicia la preparación cuando la generación finaliza con un código de retorno  $\leq 4$ . Los valores válidos son YES y NO, y el valor por omisión es YES.

Aunque establezca **prep** en NO, puede preparar código más tarde. Para obtener detalles, consulte el apartado *Invocar un plan de construcción después de la generación*.

Tenga en cuenta lo siguiente:

- Cuando se genera código en un directorio, EGL escribe mensajes de preparación en el directorio especificado en la opción **genDirectory** del descriptor de construcción, en el archivo de resultados
- Cuando se genera código en un proyecto (opción **genProject**), la opción **prep** no tienen ningún efecto y la preparación se produce en cualquiera de estas dos situaciones:
  - Siempre que se reconstruye el proyecto
  - Siempre que se generan los archivos fuente; pero sólo si ha marcado la preferencia del entorno de trabajo **Realizar construcción automáticamente al modificar el recurso**

Si desea personalizar el plan de construcción generado, haga lo siguiente:

- Establezca la opción **prep** en NO
- Establezca la opción **buildPlan** en YES (el valor por omisión)
- Genere la salida
- Personalice el plan de construcción
- Invoque el plan de construcción, como se describe en el apartado *buildPlan*

### Conceptos relacionados

“Archivo de resultados” en la página 328

### Tareas relacionadas

“Invocar un plan de construcción tras la generación” en la página 334

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“buildPlan” en la página 385

“Salida generada (referencia)” en la página 530

“genDirectory” en la página 391

“genProject” en la página 393

## resourceAssociations

La opción **resourceAssociations** del descriptor de construcción contiene el nombre de un componente de asociaciones de recurso, que relaciona componentes de registro con archivos y colas de las plataformas destino. Esta opción no es obligatoria para la generación y no hay ningún valor por omisión disponible.

### Conceptos relacionados

“Asociaciones de recursos y tipos de archivo” en la página 304

### Tareas relacionadas

“Añadir componente asociaciones recursos a archivo de construcción EGL” en la página 309

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“Elementos de asociación” en la página 375

“Referencias cruzadas de tipo de registro y tipo de archivo” en la página 737

## sessionBeanID

La opción **sessionBeanID** del descriptor de construcción identifica el nombre de un elemento de sesión existente en el descriptor de despliegue J2EE. Las entradas del entorno se colocan en el elemento de sesión cuando el usuario actúa de la forma siguiente:

- Genera un programa para una plataforma Java (estableciendo **system** en AIX, WIN o USS)
- Genera en un proyecto EJB (estableciendo **genProject** en un proyecto EJB)
- Solicita la generación de propiedades de entorno (estableciendo **genProperties** en GLOBAL o PROGRAM)

La opción **sessionBeanID** resulta de utilidad en el siguiente caso:

1. El usuario genera una envoltura Java, junto con un bean de sesión EJB. En el descriptor de despliegue de proyecto EJB (archivo `ejb-jar.xml`), EGL crea un elemento de sesión, sin entradas de entorno.

Tanto el bean de sesión EJB como el elemento de sesión se denominan de la forma siguiente:

*NombreprogramaBeanEJB*

*Nombreprograma* es el nombre del programa de tiempo de ejecución que recibe datos por medio del bean de sesión EJB. La primera letra del nombre es mayúscula, y las demás minúsculas.

En este ejemplo, el nombre del programa es `ProgramA`, y el nombre del elemento de sesión y del bean de sesión EJB es `ProgramaEJBBean`.

- Después de generar el bean de sesión EJB, el usuario genera el programa Java propiamente dicho. Dado que la opción **genProperties** del descriptor de construcción está establecida en YES, EGL genera entradas de entorno J2EE en el descriptor de despliegue, en el elemento de sesión establecido en el paso 1.
- El usuario genera ProgramB, que es un programa Java utilizado como clase de ayuda para ProgramA. Los valores de **system** y **genProject** son los mismos que los utilizados en el paso 2; asimismo, debe generar entradas de entorno y establecer **sessionBeanID** en el nombre del elemento de sesión.

La utilización de **sessionBeanID** hace que EGL coloque las entradas de entorno correspondientes al segundo programa en el elemento de sesión creado en el paso 2; específicamente, en el elemento de sesión ProgramaEJBBean.

En el componente del descriptor de despliegue que figura a continuación, EGL ha creado las entradas de entorno **vgj.nls.code** y **vgj.nls.number.decimal** durante el paso 2, cuando se generó ProgramA; pero la entrada **vgj.jdbc.default.database** sólo la utiliza ProgramB y se creó durante el paso 3:

```
<ejb-jar id="ejb-jar_ID">
 <display-name>EJBTest</display-name>
 <enterprise-beans>
 <session id="ProgramaEJBBean">
 <ejb-name>ProgramaEJBBean</ejb-name>
 <home>test.ProgramaEJBHome</home>
 <remote>test.ProgramaEJB</remote>
 <ejb-class>test.ProgramaEJBBean</ejb-class>
 <session-type>Stateful</session-type>
 <transaction-type>Container</transaction-type>
 <env-entry>
 <env-entry-name>vgj.nls.code</env-entry-name>
 <env-entry-type>java.lang.String</env-entry-type>
 <env-entry-value>ENU</env-entry-value>
 </env-entry>
 <env-entry>
 <env-entry-name>vgj.nls.number.decimal</env-entry-name>
 <env-entry-type>java.lang.String</env-entry-type>
 <env-entry-value>.</env-entry-value>
 </env-entry>
 <env-entry>
 <env-entry-name>vgj.jdbc.default.database</env-entry-name>
 <env-entry-type>java.lang.String</env-entry-type>
 <env-entry-value>jdbc/Sample</env-entry-value>
 </env-entry>
 </session>
 </enterprise-beans>
</ejb-jar>
```

Un elemento de sesión debe estar ya en el descriptor de despliegue para poder añadir entradas de entorno. Dado que los elementos de sesión se crean durante la generación de la envoltura Java, es aconsejable generar la envoltura Java antes de generar los programas relacionados.

En los casos siguientes, se genera un programa en un proyecto EJB, pero las entradas de entorno se colocan en un archivo de entorno J2EE en lugar de en el descriptor de despliegue:

- Se establece **sessionBeanID**, pero el elemento de sesión que coincide con el valor de **sessionBeanID** no se encuentra en el descriptor de despliegue; o
- No se establece **sessionBeanID**, y el elemento de sesión que se nombra para el programa no se encuentra en el descriptor de despliegue. Esta situación se produce cuando el programa se genera antes que la envoltura.

En los proyectos EJB, un nombre de entrada de entorno (como por ejemplo **vgj.nls.code**) sólo puede aparecer una vez para cada elemento de sesión. Si ya existe una entrada de entorno, EGL actualiza el tipo y el valor de la entrada en lugar de crear una nueva.

EGL nunca suprime una entrada de entorno de un descriptor de despliegue.

No hay ningún valor por omisión disponible para **sessionBeanID**.

#### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

#### **sqlCommitControl**

La opción de descriptor de construcción **sqlCommitControl** permite la generación de una propiedad de tiempo de ejecución Java que especifica si se produce un compromiso después de cada cambio en la base de datos predeterminada.

La propiedad (`vgj.jdbc.default.database.autoCommit`) solo se genera si la opción de descriptor de construcción **genProperties** también se establece en PROGRAM o GLOBAL. Puede establecer la propiedad de tiempo de ejecución Java durante el despliegue independientemente de cuál sea su decisión durante la generación.

Los valores válidos de **sqlCommitControl** son los siguientes:

#### **NOAUTOCOMMIT**

El compromiso no es automático, el comportamiento es coherente con las versiones anteriores de EGL y la propiedad de tiempo de ejecución Java se establece en false, lo que constituye el valor predeterminado.

Para conocer los detalles acerca de las reglas de compromiso y retrotracción en este caso, consulte la sección *Unidad de trabajo lógica*.

#### **AUTOCOMMIT**

El compromiso es automático, el comportamiento es coherente con versiones anteriores del producto I4GL de Informix y la propiedad de tiempo de ejecución Java se establece en true.

#### **Conceptos relacionados**

“Opciones del descriptor de construcción” en la página 382

“Propiedades de tiempo de ejecución Java” en la página 347

#### **Consulta relacionada**

“Opciones del descriptor de construcción” en la página 382

“Base de datos por omisión” en la página 251

“genProperties” en la página 394

#### **sqlDB**

La opción **sqlDB** del descriptor de construcción especifica la base de datos por omisión utilizada por un programa Java generado que se ejecuta fuera de J2EE. El valor es un URL de conexión; por ejemplo, `jdbc:db2:MyDB`.

La opción **sqlDB** es sensible a mayúsculas y minúsculas, no tiene valor por omisión y sólo se utiliza al generar un programa Java no J2EE. La opción asigna un valor a la propiedad de entorno de ejecución Java **vgj.jdbc.default.database**, pero sólo si la opción **genProperties** está establecida en GLOBAL o PROGRAM.

Para especificar la base de datos utilizada para la validación, establezca **sqlValidationConnectionURL**.

### Conceptos relacionados

“Propiedades de tiempo de ejecución Java” en la página 347  
“Soporte de SQL” en la página 229

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382  
“genProperties” en la página 394  
“Propiedades de ejecución de Java (detalles)” en la página 540  
“sqlPassword” en la página 403  
“sqlValidationConnectionURL” en la página 403  
“sqlJDBCdriverClass”  
“validateSQLStatements” en la página 406

### sqlID

La opción **sqlID** del descriptor de construcción especifica un ID de usuario utilizado para conectarse a una base de datos durante la validación en tiempo de generación de las sentencias SQL. La base de datos se especifica estableciendo **sqlValidationConnectionURL**.

Al generar un programa Java, EGL también asigna el valor de **sqlID** a la propiedad de entorno de ejecución Java **vgj.jdbc.default.userid**. Esa propiedad identifica el ID de usuario para conectarse a la base de datos por omisión durante la ejecución, y puede especificar la base de datos por omisión en **sqlDB**.

La opción **sqlID** es sensible a mayúsculas y minúsculas y no tiene valor por omisión.

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382  
“Propiedades de ejecución de Java (detalles)” en la página 540  
“sqlDB” en la página 400  
“sqlPassword” en la página 403  
“sqlValidationConnectionURL” en la página 403  
“sqlJDBCdriverClass”  
“validateSQLStatements” en la página 406

### sqlJDBCdriverClass

La opción **sqlJDBCdriverClass** del descriptor de construcción especifica una clase de controlador para conectarse a la base de datos utilizada por EGL para validar las sentencias SQL durante la generación. La base de datos se especifica estableciendo **sqlValidationConnectionURL**. El acceso a la base de datos se realiza mediante JDBC.

En los siguientes casos, EGL también asigna el valor de **sqlJDBCdriverClass** a la propiedad de entorno de ejecución Java **vgj.jdbc.drivers.userid** en el archivo de propiedades de programa:

- **genProperties** está establecida en GLOBAL o PROGRAM
- **J2EE** está establecida en NO

No hay ningún valor por omisión disponible para la clase de controlador y el formato varía según el controlador:

- Para IBM DB2 APP DRIVER para Windows, la clase de controlador es la siguiente:

```
COM.ibm.db2.jdbc.app.DB2Driver
```

- Para IBM DB2 NET DRIVER para Windows, la clase de controlador es la siguiente:  
`COM.ibm.db2.jdbc.net.DB2Driver`
- Para IBM DB2 UNIVERSAL DRIVER para Windows, la clase de controlador es la siguiente (especificando com en minúsculas):  
`com.ibm.db2.jcc.DB2Driver`
- Para el controlador del lado del cliente ligero Oracle JDBC, la clase de controlador es la siguiente:  
`oracle.jdbc.driver.OracleDriver`
- Para el controlador IBM Informix JDBC, la clase de controlador es la siguiente:  
`com.informix.jdbc.IfxDriver`

Para otras clases de controlador, consulte la documentación de cada uno de ellos.

Para especificar más de una clase de controlador, separe cada nombre de clase del siguiente mediante un signo de dos puntos (:). Puede que esto le interese si un programa Java efectúa una llamada local a otro, pero accede a un sistema de gestión de bases de datos diferente.

#### Consulta relacionada

- “Opciones del descriptor de construcción” en la página 382
- “Informix y EGL” en la página 252
- “sqlDB” en la página 400
- “sqlID” en la página 401
- “sqlPassword” en la página 403
- “sqlValidationConnectionURL” en la página 403
- “validateSQLStatements” en la página 406

#### sqlJNDIName

La opción **sqlJNDIName** del descriptor de construcción especifica la base de datos por omisión utilizada por un programa Java generado que se ejecuta en J2EE. El valor es el nombre al que está enlazado el origen de datos en el registro JNDI; por ejemplo, jdbc/MyDB.

La opción **sqlJNDIName** es sensible a mayúsculas y minúsculas, no tiene valor por omisión y sólo se utiliza al generar un programa Java para J2EE. La opción asigna un valor a la propiedad de entorno de ejecución Java **vgj.jdbc.default.database**, pero sólo si la opción **genProperties** está establecida en GLOBAL o PROGRAM.

Para especificar la base de datos utilizada para la validación , establezca **sqlValidationConnectionURL**.

#### Conceptos relacionados

- “Propiedades de tiempo de ejecución Java” en la página 347
- “Soporte de SQL” en la página 229

#### Consulta relacionada

- “Opciones del descriptor de construcción” en la página 382
- “genProperties” en la página 394
- “Propiedades de ejecución de Java (detalles)” en la página 540
- “sqlPassword” en la página 403
- “sqlValidationConnectionURL” en la página 403
- “sqlJDBCdriverClass” en la página 401
- “validateSQLStatements” en la página 406

## sqlPassword

La opción **sqlPassword** del descriptor de construcción especifica una contraseña utilizada para conectarse a una base de datos durante la validación en tiempo de generación de las sentencias SQL. La base de datos se especifica estableciendo **sqlValidationConnectionURL**.

Al generar un programa Java, EGL también asigna el valor de **sqlPassword** a la propiedad de entorno de ejecución Java **vgj.jdbc.default.password**. Esa propiedad identifica la contraseña para conectarse a la base de datos por omisión durante la ejecución, y puede especificar la base de datos por omisión en **sqlDB**.

La opción **sqlPassword** es sensible a mayúsculas y minúsculas y no tiene valor por omisión.

### Conceptos relacionados

“Propiedades de tiempo de ejecución Java” en la página 347

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“Propiedades de ejecución de Java (detalles)” en la página 540

“sqlDB” en la página 400

“sqlID” en la página 401

“sqlValidationConnectionURL”

“sqlJDBCdriverClass” en la página 401

“validateSQLStatements” en la página 406

## sqlValidationConnectionURL

La opción **sqlValidationConnectionURL** del descriptor de construcción especifica un URL para conectarse a la base de datos utilizada por EGL para validar las sentencias SQL durante la generación. El acceso a la base de datos se realiza mediante JDBC.

No hay ningún valor por omisión disponible para el URL y el formato varía según el controlador:

- Para IBM DB2 APP DRIVER para Windows, el URL es el siguiente:

```
jdbc:db2:nombreBaseDatos
```

*nombreBaseDatos*

Nombre de la base de datos

- Para el controlador del lado del cliente ligero Oracle JDBC, el URL varía en función de la ubicación de la base de datos. Si la base de datos es local con respecto a la máquina, el URL es el siguiente:

```
jdbc:oracle:thin:nombreBaseDatos
```

Si la base de datos se encuentra en un servidor remoto, el URL es el siguiente:

```
jdbc:oracle:thin:@sistemaprincipal:puerto:nombreBaseDatos
```

*sistemaprincipal*

Nombre de sistema principal del servidor de bases de datos

*puerto*

Número de puerto

*nombreBaseDatos*

Nombre de la base de datos

- Para el controlador IBM Informix JDBC, el URL es el siguiente (en una sola línea):

```
jdbc:informix-sqli://sistemaprincipal:puerto
/nombreBaseDatos:informixserver=nombreServidor;
user=nombreUsuario;password=Contraseña
```

*sistemaprincipal*

Nombre de la máquina en la que reside el servidor de bases de datos

*puerto*

Número de puerto

*nombreBaseDatos*

Nombre de la base de datos

*nombreServidor*

Nombre del servidor de bases de datos

*nombreUsuario*

ID de usuario de Informix

*Contraseña*

Contraseña asociada con el ID de usuario

- Para otros controladores, consulte la documentación de cada uno de ellos.

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“Informix y EGL” en la página 252

“sqlDB” en la página 400

“sqlID” en la página 401

“sqlPassword” en la página 403

“sqlJDBCdriverClass” en la página 401

“validateSQLStatements” en la página 406

### system

La opción **system** del descriptor de construcción especifica la plataforma destino de la generación. esta opción es obligatoria; no hay ningún valor por misión disponible. Los valores válidos son los siguientes:

#### AIX

Indica que la generación produce un programa Java que puede ejecutarse en AIX

#### ISERIESJ

Indica que la generación produce un programa Java que puede ejecutarse en iSeries

#### LINUX

Indica que la generación produce un programa Java que puede ejecutarse en Linux (con un procesador Intel)

#### USS

Indica que la generación produce un programa Java que puede ejecutarse en z/OS UNIX System Services

#### WIN

Indica que la generación produce un programa Java que puede ejecutarse en Windows 2000/NT/XP

### Conceptos relacionados

“Salida generada” en la página 529

“Componente de opciones de enlace” en la página 311

“Configuraciones de tiempo de ejecución” en la página 9

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“Elemento callLink” en la página 407

“Salida generada (referencia)” en la página 530

“Informix y EGL” en la página 252

### targetNLS

La opción **targetNLS** del descriptor de construcción especifica el código de idioma nacional utilizado para identificar los mensajes de tiempo de ejecución.

La tabla siguiente lista todos los idiomas soportados. La página de códigos del idioma que especifique debe cargarse en la plataforma destino.

Código	Idiomas
CHS	Chino simplificado
CHT	Chino tradicional
DES	Alemán de Suiza
DEU	Alemán
ENP	Inglés en mayúsculas (no soportado en Windows 2000, Windows NT ni z/OS UNIX System Services)
ENU	Inglés de Estados Unidos
ESP	Español
FRA	Francés
ITA	Italiano
JPN	Japonés
KOR	Coreano
PTB	Portugués de Brasil

EGL determina si el entorno local Java de la máquina de desarrollo está asociado con uno de los idiomas soportados. Si la respuesta es “yes”, el valor por omisión de **targetNLS** será el idioma soportado. De lo contrario, **targetNLS** no tendrá valor por omisión.

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

### VAGCompatibility

La opción **VAGCompatibility** del descriptor de construcción indica si el proceso de generación permite la utilización de la sintaxis de programa especial, como se describe en el apartado *Compatibilidad con VisualAge Generator*. Los valores válidos son *no* y *yes*.

El valor de la preferencia EGL **VAGCompatibility** determina el valor por omisión del descriptor de construcción. Si está generando en el SDK de EGL, no hay preferencias disponibles y el valor por omisión de **VAGCompatibility** es *no*.

Especifique *yes* sólo si el programa o el PageHandler utilizan la sintaxis especial.

### Conceptos relacionados

“Componente descriptor de construcción” en la página 293

“Compatibilidad con VisualAge Generator” en la página 439

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

### validateSQLStatements

La opción **validateSQLStatements** del descriptor de construcción indica si las sentencias SQL se validan con una base de datos. La utilización satisfactoria de **validateSQLStatements** requiere la especificación de la opción **sqlValidationConnectionURL** y, en la mayoría de los casos, de otras opciones que empiecen con las letras **sql**, como se indica más adelante.

Los valores válidos son YES y NO, y el valor por omisión es NO. La validación de sentencias SQL aumenta el tiempo necesario para generar el código.

Cuando se solicita la validación de SQL, el gestor de bases de datos al que se accede desde la plataforma de generación prepara las sentencias SQL dinámicamente.

La validación de sentencias SQL tiene estas restricciones:

- No es posible realizar la validación de sentencias SQL que utilizan SQL dinámico y se basan en registros SQL
- El proceso de validación puede indicar errores encontrados por el gestor de bases de datos en el entorno de generación, pero que el gestor de bases de datos no encontrará en la plataforma destino
- La validación sólo se produce si el controlador JDBC da soporte a la validación de sentencias SQL prepare y (en algunos casos) sólo si ha configurado el controlador para realizar la validación; para obtener detalles, consulte la documentación del controlador JDBC

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“sqlID” en la página 401

“sqlPassword” en la página 403

“sqlValidationConnectionURL” en la página 403

“sqlJDBCClass” en la página 401

---

## Scripts de construcción

### Opciones necesarias en los scripts de construcción de EGL

En los scripts de construcción EGL, determinadas opciones de preparación son obligatorias si utiliza DB2 UDB.

#### Opciones obligatorias para el precompilador de DB2

Las opciones siguientes son obligatorias para el uso de DB2 y están incluidas en el script de construcción fdaptcl:

- HOST(COB2)
- APOSTSQL
- QUOTE

---

## Elemento callLink

El elemento callLink de un componente de opciones de enlace especifica el tipo de enlace utilizado en una llamada. Cada elemento incluye estas propiedades:

- pgmName
- type

El valor de la propiedad **type** determina qué propiedades adicionales están disponibles, como se muestra en las siguientes secciones:

- “Si el tipo de callLink es localCall (valor por omisión)”
- “Si el tipo de callLink es remoteCall”
- “Si el tipo de callLink es.ejbCall” en la página 408

### Si el tipo de callLink es localCall (valor por omisión)

Establezca la propiedad **type** en localCall si está generando un programa Java que llama a un programa Java generado que reside en la misma hebra. En este caso, el middleware EGL no se utiliza, y las siguientes propiedades son relevantes para un elemento callLink en el que **pgmName** identifique el programa llamado:

- “Propiedad alias del elemento callLink” en la página 409
- “Propiedad package del elemento callLink” en la página 415
- “Propiedad pgmName del elemento callLink” en la página 417
- “Propiedad type del elemento callLink” en la página 423

No es necesario especificar un elemento callLink para la llamada si el programa llamado está en el mismo paquete que el llamador y si se cumple alguna de las siguientes condiciones:

- No especifica un nombre externo para el programa llamado; o
- El nombre externo del programa llamado es idéntico al nombre de componente de ese programa.

El valor de **type** no puede ser localCall al generar una envoltura Java.

### Si el tipo de callLink es remoteCall

Establezca la propiedad **type** en remoteCall si está generando un programa o envoltura Java y el código Java llama a un programa que se ejecuta en una hebra diferente. La llamada no se realiza por medio de un bean de sesión EJB generado. En este caso, el middleware EGL se utiliza, y las siguientes propiedades son relevantes para un elemento callLink en el que **pgmName** identifique el programa llamado:

- “Propiedad alias del elemento callLink” en la página 409
- “Propiedad conversionTable del elemento callLink” en la página 409
- “Propiedad location del elemento callLink” en la página 413
- “Propiedad package del elemento callLink” en la página 415 (utilizada sólo si el código generado llama a un programa Java almacenado en otro paquete)
- “Propiedad pgmName del elemento callLink” en la página 417
- “Propiedad remoteBind del elemento callLink” en la página 418
- “Propiedad remoteComType del elemento callLink” en la página 419
- “Propiedad remotePgmType del elemento callLink” en la página 421
- “Propiedad serverID del elemento callLink” en la página 422
- “Propiedad type del elemento callLink” en la página 423

## Si el tipo de callLink es.ejbCall

establezca la propiedad **type** en `ejbCall` cuando sea necesario un elemento `callLink` para manejar alguna de las siguientes situaciones:

- Está generando una envoltura Java y tiene intención de llamar al programa generado relacionado por medio de un bean de sesión EJB generado
- Está generando un programa Java y tiene intención de llamar a otro programa generado por medio de un bean de sesión EJB generado

En este caso, el middleware EGL se utiliza, y las siguientes propiedades son relevantes para un elemento `callLink` en el que **pgmName** identifique el programa llamado:

- “Propiedad alias del elemento `callLink`” en la página 409
- “Propiedad `conversionTable` del elemento `callLink`” en la página 409
- “Propiedad `location` del elemento `callLink`” en la página 413
- “Propiedad `package` del elemento `callLink`” en la página 415 (utilizada sólo si el código Java generado llama a un programa Java almacenado en un paquete que no es aquél en el que reside el bean de sesión EJB)
- “Propiedad `parmForm` del elemento `callLink`” en la página 416 (utilizada sólo si el código Java generado llama a un programa que se ejecuta en CICS)
- “Propiedad `pgmName` del elemento `callLink`” en la página 417
- “`providerURL` en elemento `callLink`” en la página 417
- “Propiedad `remoteBind` del elemento `callLink`” en la página 418
- “Propiedad `remoteComType` del elemento `callLink`” en la página 419
- “Propiedad `remotePgmType` del elemento `callLink`” en la página 421
- “Propiedad `serverID` del elemento `callLink`” en la página 422
- “Propiedad `type` del elemento `callLink`” en la página 423

### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

“Configuraciones de tiempo de ejecución” en la página 9

### Tareas relacionadas

“Editar el elemento `callLink` de un componente de opciones de enlace” en la página 314

### Consulta relacionada

“Propiedad alias del elemento `callLink`” en la página 409

“Propiedad `conversionTable` del elemento `callLink`” en la página 409

“Propiedad `linkType` del elemento `callLink`” en la página 412

“Propiedad `location` del elemento `callLink`” en la página 413

“Propiedad `package` del elemento `callLink`” en la página 415

“Propiedad `pgmName` del elemento `callLink`” en la página 417

“`providerURL` en elemento `callLink`” en la página 417

“Propiedad `remoteBind` del elemento `callLink`” en la página 418

“Propiedad `remoteComType` del elemento `callLink`” en la página 419

“Propiedad `remotePgmType` del elemento `callLink`” en la página 421

“Propiedad `serverID` del elemento `callLink`” en la página 422

“Propiedad `type` del elemento `callLink`” en la página 423

## Propiedad alias del elemento callLink

El componente de opciones de enlace, elemento callLink, propiedad **alias** especifica el nombre del programa identificado en la propiedad **pgmName**. La propiedad sólo es relevante cuando **pgmName** hace referencia a un programa al que llama el programa que se genera.

El valor de esta propiedad debe coincidir con el alias (si existe) que ha especificado al declarar el programa. Si no ha especificado un alias al declarar el programa, establezca la propiedad del elemento callLink **alias** en el nombre del componente de programa o no establezca la propiedad en absoluto.

### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

### Tareas relacionadas

“Editar el elemento callLink de un componente de opciones de enlace” en la página 314

### Consulta relacionada

“Elemento callLink” en la página 407

“Propiedad pgmName del elemento callLink” en la página 417

## Propiedad conversionTable del elemento callLink

El componente de opciones de enlace, callLink, propiedad **conversionTable** especifica el nombre de la tabla de conversión utilizada para convertir datos de una llamada. La propiedad sólo es relevante cuando **pgmName** identifica un programa al que llama el programa o envoltura generados.

Losiguiente:

- Cuando la llamada se realiza a un programa no Java, se produce una conversión por omisión de acuerdo con el juego de caracteres (ASCII o EBCDIC) utilizado en la plataforma llamante. Debe especificar un valor para **conversionTable** en el caso siguiente:
  - El llamador es código Java y se encuentra en una máquina que da soporte a un juego de caracteres (EBCDIC o ASCII); y
  - El programa llamado no es Java y se encuentra en una máquina que da soporte al otro juego de caracteres.
- El intento de especificar una tabla de conversión no tiene ningún efecto cuando el código Java generado por EGL llama a un programa Java, excepto en el caso de texto bidireccional.
- La propiedad **conversionTable** sólo está disponible si el valor de la propiedad **type** es **ejbCall** o **remoteCall**.

Seleccione uno de los siguientes valores:

*nombre de tabla de conversión*

El llamador utiliza la tabla de conversión especificada. Para obtener una lista de tablas, consulte el apartado *Conversión de datos*.

- \* Se utiliza la tabla de conversión por omisión. La tabla seleccionada se basa en el entorno local de la máquina cliente (si el cliente se ejecuta en un servidor Web Application Server) o en el entorno local de ese servidor. Si se encuentra un entorno local no reconocido, se presupone el Inglés.

Para obtener una lista de tablas, consulte el apartado *Conversión de datos*.

### **programControlled**

El llamador utiliza el nombre de tabla de conversión que se encuentra en el elemento del sistema sysVar.callConversionTable durante la ejecución. Si sysVar.callConversionTable contiene blancos, no se realiza ninguna conversión.

### **Conceptos relacionados**

“Componente de opciones de enlace” en la página 311

### **Tareas relacionadas**

“Editar el elemento callLink de un componente de opciones de enlace” en la página 314

### **Consulta relacionada**

“Texto de idioma bidireccional” en la página 470

“Elemento callLink” en la página 407

“Conversión de datos” en la página 467

“Propiedad pgmName del elemento callLink” en la página 417

“convert()” en la página 897

“targetNLS” en la página 405

“Propiedad type del elemento callLink” en la página 423

## **Propiedad ctgKeyStore del elemento callLink**

El componente de opciones de enlace, elemento callLink, propiedad **ctgKeyStore** es el nombre del almacén de claves generado con la herramienta Java keytool.exe o con la herramienta de Pasarela de transacción CICS IKEYMAN. Esta propiedad es obligatoria si el valor de la propiedad **remoteComType** está establecido en CICSSL.

### **Conceptos relacionados**

“Componente de opciones de enlace” en la página 311

### **Consulta relacionada**

“Elemento callLink” en la página 407

“Propiedad ctgKeyStorePassword del elemento callLink”

“Propiedad remoteComType del elemento callLink” en la página 419

## **Propiedad ctgKeyStorePassword del elemento callLink**

El componente de opciones de enlace, elemento callLink, propiedad **ctgKeyStorePassword** es la contraseña utilizada al generar el almacén de claves.

### **Conceptos relacionados**

“Componente de opciones de enlace” en la página 311

### **Consulta relacionada**

“Elemento callLink” en la página 407

“Propiedad ctgKeyStore del elemento callLink”

“Propiedad remoteComType del elemento callLink” en la página 419

## **Propiedad ctgLocation del elemento callLink**

El componente de opciones de enlace, elemento callLink, propiedad **ctgLocation** es el URL utilizado para acceder a un servidor de Pasarela de transacción CICS (CTG), que se utiliza si el valor de la propiedad **remoteComType** es CICSECI o CICSSL. Especifique el puerto relacionado estableciendo la propiedad **ctgPort**.

### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

### Consulta relacionada

“Elemento callLink” en la página 407

“Propiedad remoteComType del elemento callLink” en la página 419

## Propiedad ctgPort del elemento callLink

El componente de opciones de enlace, elemento callLink, propiedad **ctgPort** es el puerto utilizado para acceder a un servidor de Pasarela de transacción CICS (CTG), que se utiliza si el valor de la propiedad **remoteComType** es CICSECI o CICSSSL. Especifique el URL relacionado estableciendo la propiedad **ctgLocation**.

En el caso de CICSSSL, el valor de **ctgPort** es el puerto TCP/IP en el que un escuchador CTG JSSE está a la escucha de las peticiones; y, si no se especifica **ctgPort**, se utiliza el puerto por omisión de CTG 8050.

### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

### Consulta relacionada

“Elemento callLink” en la página 407

“Propiedad ctgLocation del elemento callLink” en la página 410

“Propiedad remoteComType del elemento callLink” en la página 419

## Propiedad javaWrapper del elemento callLink

El componente de opciones de enlace, elemento **callLink**, propiedad **javaWrapper** indica si se permite la generación de clases de envoltura Java que pueden invocar el programa que se genera.

Los valores válidos son los siguientes:

### No (el valor por omisión)

No se permite la generación de clases de envoltura Java.

### Yes

Permite que se produzca la generación. La generación sólo se produce si la opción **enableJavaWrapperGen** del descriptor de construcción está establecida en **yes** o **only**.

Su elección con respecto a la propiedad **javaWrapper** sólo tiene efecto al configurar una llamada remota, como ocurre cuando el valor de **callLink**, propiedad **type** es **remoteCall**. Por el contrario, si configura una llamada al programa por medio de un EJB, el valor de **javaWrapper** es siempre **yes**; y, si configura una llamada local, el valor de **javaWrapper** es siempre **no**.

Si la generación se realiza en el entorno de trabajo o desde la interfaz de proceso por lotes del entorno de trabajo, la opción **genProject** del descriptor de construcción identifica el proyecto que recibe las clases. Si no se especifica **genProject** (o si la generación se realiza en el SDK de EGL), las clases de envoltura se colocan en el directorio especificado en la opción **genDirectory** del descriptor de construcción.

### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

### Consulta relacionada

“Elemento callLink” en la página 407

“genDirectory” en la página 391

“genProject” en la página 393

## Propiedad linkType del elemento callLink

El componente de opciones de enlace, elemento callLink, propiedad **linkType** especifica el tipo de enlace cuando el valor de la propiedad **type** es localCall.

Si está generando un programa Java, **linkType** es relevante cuando la propiedad **pgmName** hace referencia a un programa al que llama el programa que se genera. Si está generando una envoltura Java, la propiedad **type** debe ser remoteCall o.ejbCall, y **linkType** no está disponible.

Seleccione un valor en esta lista:

### DYNAMIC

Especifica que la llamada se realiza a un programa Java de la misma hebra. DYNAMIC es el valor por omisión .

### STATIC

STATIC es equivalente a DYNAMIC a menos que utilice Rational Application Developer para iSeries o Rational Application Developer para z/OS..

### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

### Tareas relacionadas

“Editar el elemento callLink de un componente de opciones de enlace” en la página 314

### Consulta relacionada

“Elemento callLink” en la página 407

“Propiedad pgmName del elemento callLink” en la página 417

“Propiedad type del elemento callLink” en la página 423

## Propiedad library del elemento callLink

El componente de opciones de enlace, elemento callLink, propiedad **library** especifica la DLL o biblioteca que contiene el programa llamado cuando el valor de la propiedad **type** es.ejbCall o remoteCall:

- Si el programa Java generado por EGL está llamando a un programa remoto no generado por EGL en iSeries (por ejemplo, un programa de servicio C o C++), el programa llamado pertenece a una biblioteca de iSeries, y la propiedad **library** hace referencia al nombre del programa que contiene el punto de entrada al que debe llamarse. Establezca las demás propiedades de callLink como se indica a continuación:
  - Establezca la propiedad **pgmName** en el nombre del punto de entrada
  - Establezca la propiedad **remoteComType** en direct o distinct
  - Establezca la propiedad **remotePgmType** en externallyDefined
  - Establezca la propiedad **location** en el nombre de la biblioteca iSeries

- De lo contrario, si el programa de llamada es un programa Java generado por EGL, no en iSeries, la propiedad **library** hace referencia al nombre de una DLL que contiene un punto de entrada al que debe llamarse localmente como programa nativo. El punto de entrada se identifica en la propiedad **pgmName**; pero sólo es necesario especificar la propiedad **library** si los nombres del punto de entrada y la DLL son diferentes.

Para llamar a un DLL nativa, establezca las demás propiedades de callLink del siguiente modo:

- Establezca la propiedad **remoteComType** en direct
- Establezca la propiedad **remotePgmType** en externallyDefined
- Establezca la propiedad **type** en remoteCall, ya que el middleware de EGL se utiliza aunque la DLL se invoque desde la máquina en la que se ejecuta el programa Java.

#### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

#### Consulta relacionada

“Elemento callLink” en la página 407

## Propiedad location del elemento callLink

El componente de opciones de enlace, elemento callLink, propiedad **location** especifica cómo se determina durante la ejecución la ubicación de un programa llamado. La propiedad **location** es aplicable en la siguiente situación:

- El valor de la propiedad **type** es.ejbCall o remoteCall;
- El valor de la propiedad **remoteComType** es JAVA400, CICSECI, CICSSSL, CICSJ2C o TCPIP; y
- Se aplica una de las siguientes condiciones:
  - Si está generando un programa Java, la propiedad **pgmName** hace referencia a un programa al que llama el programa que se genera
  - Si está generando una envoltura Java, **pgmName** hace referencia a un programa al que se llama por medio de la envoltura Java.

Seleccione un valor de esta lista:

#### **programControlled**

Especifica que la ubicación del programa llamado se obtiene de la función de sistema sysVar.remoteSystemID cuando se produce la llamada.

#### *nombre de sistema*

Especifica la ubicación en la que reside el programa llamado.

Si está generando un programa o envoltura Java, el significado de esta propiedad depende de la propiedad **remoteComType**:

- Si el valor de **remoteComType** es JAVA400, **location** hace referencia al identificador del sistema iSeries
- Si el valor de **remoteComType** es CICSECI o CICSSSL, **location** hace referencia al identificador del sistema CICS
- Si el valor de **remoteComType** es CICSJ2C, **location** hace referencia al nombre JNDI del objeto ConnectionFactory establecido para la transacción CICS invocada por la llamada. Ese objeto ConnectionFactory se establece al configurar el servidor J2EE, según se describe en el apartado *Configurar el servidor J2EE para llamadas CICSJ2C*. Por convenio, el nombre del objeto ConnectionFactory empieza por eis/, como en el ejemplo siguiente:

eis/CICS1

- Si el valor de **remoteComType** es TCPIP, **location** hace referencia al nombre de sistema principal TCP/IP, y no existe valor por omisión
- Si se cumple la totalidad de las siguientes condiciones, **location** hace referencia a la biblioteca del programa llamado:
  - El programa llamado es un programa Java generado por EGL que se ejecuta localmente en iSeries
  - El valor de **remoteComType** es DIRECT o DISTINCT
  - El valor de **remotePgmType** es EXTERNALLYDEFINED

#### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

#### Tareas relacionadas

“Editar el elemento callLink de un componente de opciones de enlace” en la página 314

“Configuración del servidor J2EE para llamadas CICSJ2C” en la página 358

#### Consulta relacionada

“Elemento callLink” en la página 407

“Propiedad pgmName del elemento callLink” en la página 417

“Propiedad remoteComType del elemento callLink” en la página 419

“Propiedad type del elemento callLink” en la página 423

## Propiedad **luwControl** del elemento **callLink**

El componente de opciones de enlace, elemento **callLink**, propiedad **luwControl** especifica si el programa llamado o llamador controlan la unidad de trabajo. Esta propiedad sólo es aplicable en la siguiente situación:

- El valor de la propiedad **type** es **remoteCall**; y
- Está generando un programa o envoltura Java
  - Si está generando un programa Java, la propiedad **pgmName** hace referencia a un programa basado en CICS llamado por el programa que se genera.
  - Si está generando una envoltura Java, **pgmName** hace referencia a un programa basado en CICS al que se llama por medio de la envoltura Java.

Seleccione uno de los siguientes valores:

#### CLIENT

Especifica que la unidad de trabajo está bajo el control del llamador. Las actualizaciones realizadas por el programa llamado no se comprometen ni retrotraen hasta que el llamador solicita el compromiso o la retrotracción. Si el programa llamado emite un mandato **commit** o **rollback**, se produce un error de ejecución.

CLIENT es el valor por omisión, a menos que la plataforma en la que reside el programa llamado no dé soporte a una unidad de trabajo controlada por el llamador.

CLIENT está disponible si el llamador es una envoltura o programa Java que comunica con un programa COBOL basado en iSeries mediante IBM Toolbox para Java. En este caso, el valor de **remoteComType** para la llamada es JAVA400.

#### SERVER

Especifica que la unidad de trabajo iniciada por el programa llamado es

independiente de las unidades de trabajo controladas por el programa llamante. En el programa llamado, se aplican estas normas:

- El primer cambio realizado en un recuso recuperable inicia una unidad de trabajo
- La utilización de las funciones de sistema `sysLib.commit` y `sysLib.rollback` es válida

En una llamada desde código Java generado por EGL a un programa COBOL de VisualAge Generator , se emite automáticamente un compromiso (o una retrotracción en el caso de una finalización anómala) cuando el programa llamado efectúa el retorno. Ese mandato sólo afecta a los cambios efectuados por el programa llamado.

Cuando la propiedad **type** es `ejbCall`, el comportamiento de ejecución es el descrito para `SERVER`.

#### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

“Unidad lógica de trabajo” en la página 307

#### Tareas relacionadas

“Editar el elemento `callLink` de un componente de opciones de enlace” en la página 314

#### Consulta relacionada

“Elemento `callLink`” en la página 407

“`commit()`” en la página 893

“`rollback()`” en la página 905

“Propiedad `pgmName` del elemento `callLink`” en la página 417

“Propiedad `type` del elemento `callLink`” en la página 423

## Propiedad `package` del elemento `callLink`

El componente de opciones de enlace, elemento `callLink`, propiedad **package** identifica el paquete Java en el que reside el programa Java llamado. La propiedad resulta de utilidad si la propiedad **type** es `ejbcall`, `localCall` o `remoteCall`.

Si está generando un programa Java, **package** es relevante cuando la propiedad **pgmName** hace referencia a un programa llamado por el programa que se genera. Si está generando una envoltura Java, **package** es relevante cuando la propiedad **pgmName** hace referencia al programa llamado por medio de la envoltura Java.

Si no se especifica la propiedad **package**, se presupone que el programa llamado se encuentra en el mismo paquete que el llamador.

El nombre de paquete que se utiliza en los programas Java generados es el nombre de paquete del programa EGL, pero en minúsculas; y cuando EGL genera salida del elemento `callLink`, el valor **package** se cambia (si es necesario) a minúsculas.

#### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

#### Tareas relacionadas

“Editar el elemento `callLink` de un componente de opciones de enlace” en la página 314

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

#### Consulta relacionada

“Elemento `callLink`” en la página 407

“Propiedad pgmName del elemento callLink” en la página 417

“Propiedad type del elemento callLink” en la página 423

## Propiedad parmForm del elemento callLink

El componente de opciones de enlace, elemento callLink, propiedad **parmForm** especifica el formato de los parámetros de llamada.

Si está generando un programa Java, **parmForm** es aplicable en esta situación:

- La propiedad **pgmName** hace referencia a un programa basado en CICS llamado por el programa que se genera; y
- La propiedad **type** es **ejbCall** o **remoteCall**; en cualquier caso, los valores válidos de **parmForm** (como se describe más adelante) son **COMMDATA** (valor por omisión) y **COMMPTR**

Si está generando una envoltura Java, **parmForm** es aplicable en este caso:

- La propiedad **pgmName** hace referencia a un programa COBOL generado llamado por medio de la envoltura Java; y
- La propiedad **type** es **ejbCall** o **remoteCall**; en cualquier caso, los valores válidos de **parmForm** (como se describe más adelante) son **COMMDATA** (valor por omisión) o **COMMPTR**

Seleccione un valor en esta lista:

### COMMDATA

Especifica que el llamador coloca los datos comerciales (en lugar de punteros a los datos) en la **COMMAREA**.

Cada valor de argumento se traslada al almacenamiento intermedio contiguo al valor anterior independientemente de la alineación de límite.

**COMMDATA** es el valor por omisión si la propiedad **type** es **ejbCall** o **remoteCall**.

### COMMPTR

Especifica que el llamador actúa de la forma siguiente:

- Coloca una serie de punteros de 4 bytes en la **COMMAREA**, un puntero por argumento pasado
- Establece el bit de orden superior del último puntero en 1

**COMMPTR** es el valor por omisión si la propiedad **type** es **localCall**.

### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

### Tareas relacionadas

“Editar el elemento callLink de un componente de opciones de enlace” en la página 314

### Consulta relacionada

“Elemento callLink” en la página 407

“Propiedad linkType del elemento callLink” en la página 412

“Propiedad parmForm del elemento callLink”

“Propiedad pgmName del elemento callLink” en la página 417

“Propiedad type del elemento callLink” en la página 423

## Propiedad `pgmName` del elemento `callLink`

El componente de opciones de enlace, elemento `callLink`, propiedad `pgmName` especifica el nombre del componente de programa al que el elemento `callLink` hace referencia.

Puede utilizar un asterisco (\*) como carácter de sustitución global en el nombre de programa; sin embargo, ese carácter sólo es válido como último carácter. Para obtener detalles, consulte el apartado *Componente de opciones de enlace*.

### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

### Tareas relacionadas

“Editar el elemento `callLink` de un componente de opciones de enlace” en la página 314

### Consulta relacionada

“Elemento `callLink`” en la página 407

## `providerURL` en elemento `callLink`

El componente de opciones de enlace, elemento `callLink`, propiedad `providerURL` especifica el nombre de sistema principal y el número de puerto del servidor de nombres utilizado por el programa o envoltura Java generado por EGL para localizar un bean de sesión EJB que, a su vez, llama a un programa Java generado por EGL. La propiedad debe tener el siguiente formato:

```
iio://nombreSistemaPrincipal:numeroPuerto
```

*nombreSistemaPrincipal*

La dirección IP o el nombre de sistema principal de la máquina en la que se ejecuta el servidor de nombres.

*numeroPuerto*

El número de puerto en el que el servidor de nombres está a la escucha

La propiedad `providerURL` sólo es aplicable en la siguiente situación:

- El valor de la propiedad `type` es `ejbCall`; y
- La propiedad `pgmName` hace referencia al programa llamado desde el programa o envoltura Java que se genera.

Especifique el URL entre comillas para evitar un problema con los puntos o el punto y coma que preceden al número de puerto.

Si no especifica un valor para `providerURL`, se utiliza un valor por omisión. El valor por omisión dirige un cliente EJB para buscar el servidor de nombres que se encuentre en el sistema principal local y que esté a la escucha en el puerto 900. El valor por omisión es equivalente al siguiente URL:

```
"iio://"
```

El siguiente valor de `providerURL` dirige un cliente EJB para buscar un servidor de nombres remoto denominado *bankserver.mybank.com* y que está a la escucha en el puerto 9019:

```
"iio://bankserver.mybank.com:9019"
```

El siguiente valor de la propiedad dirige un cliente EJB para buscar un servidor de nombres remoto denominado *bankserver.mybank.com* y que está a la escucha en el puerto 900:

"iiop://bankserver.mybank.com"

#### Conceptos relacionados

"Componente de opciones de enlace" en la página 311

#### Tareas relacionadas

"Editar el elemento callLink de un componente de opciones de enlace" en la página 314

"Configurar entorno de ejecución J2EE para código generado por EGL" en la página 354

#### Consulta relacionada

"Elemento callLink" en la página 407

"Propiedad pgmName del elemento callLink" en la página 417

"Propiedad type del elemento callLink" en la página 423

## Propiedad refreshScreen del elemento callLink

El componente de opciones de enlace, elemento callLink, propiedad **refreshScreen** indica si debe realizarse una renovación automática de la pantalla cuando el programa llamado devuelve el control. Los valores válidos son *yes* (el valor por omisión) y *no*.

Establezca **refreshScreen** en *no* si el llamador está en una unidad de ejecución que presenta formularios de texto en una pantalla y se produce alguna de estas situaciones:

- El programa llamado no presenta un formulario de texto; o
- El llamador escribe un formulario de texto de pantalla completa después de la llamada.

La propiedad **refreshScreen** sólo se aplica en estos casos:

- La propiedad **type** de callLink es localCall; o
- La propiedad **type** de callLink es remoteCall cuando la propiedad remoteComType es direct o distinct.

La propiedad se pasa por alto si incluye el indicador **noRefresh** en la sentencia call.

#### Consulta relacionada

"call" en la página 563

## Propiedad remoteBind del elemento callLink

El componente de opciones de enlace, elemento callLink, propiedad **remoteBind** especifica si las opciones de enlace se determinan durante la generación o durante la ejecución. Esta propiedad sólo es aplicable en la siguiente situación:

- El valor de la propiedad **type** es.ejbCall o remoteCall; y
- Está generando un programa o envoltura Java. La propiedad **pgmName** puede hacer referencia a un programa llamado por el programa que se genera, en cuyo caso la entrada hace referencia a la llamada de programa a programa. Como alternativa, la propiedad puede hacer referencia al programa generado, en cuyo caso la entrada hace referencia a la llamada de envoltura a programa.

Seleccione uno de los siguientes valores:

## GENERATION

Las opciones de enlace especificadas durante la generación se utilizan necesariamente durante la ejecución. GENERATION es el valor por omisión.

## RUNTIME

Las opciones de enlace especificadas durante la generación puede revisarse durante el despliegue. En este caso, debe incluir un archivo de propiedades de enlace en el entorno de ejecución.

EGL genera un archivo de propiedades de enlace en la siguiente situación:

- Está generando un programa o envoltura Java.
- Establece el valor de la propiedad **remoteBind** en RUNTIME;
- Genera con la opción **genProperties** del descriptor de despliegue establecida en GLOBAL o PROGRAM.

### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

“Archivo de propiedades de enlace” en la página 364

### Tareas relacionadas

“Desplegar un archivo de propiedades de enlace” en la página 364

“Editar el elemento callLink de un componente de opciones de enlace” en la página 314

### Consulta relacionada

“Elemento callLink” en la página 407

“genProperties” en la página 394

“Archivo de propiedades de enlace (detalles)” en la página 657

“Propiedad pgmName del elemento callLink” en la página 417

“Propiedad type del elemento callLink” en la página 423

## Propiedad remoteComType del elemento callLink

El componente de opciones de enlace, elemento callLink, propiedad **remoteComType** especifica el protocolo de comunicaciones utilizado en el caso siguiente:

- El valor de la propiedad **type** es **ejbCall** o **remoteCall**; y
- Está generando un programa o envoltura Java
  - Si está generando un programa Java, la propiedad **pgmName** hace referencia a un programa llamado por el programa que se genera.
  - Si está generando una envoltura Java, **pgmName** hace referencia a un programa al que se llama por medio de la envoltura Java.

Seleccione uno de los siguientes valores:

### DEBUG

Hace que el programa llamado se ejecute en el depurador de EGL, aunque el programa llamante se ejecute en un entorno de ejecución Java o en el entorno de depuración Java. Puede utilizar este valor en los siguientes casos:

- Está ejecutando un programa Java que utiliza una envoltura Java EGL para llamar a un programa escrito con EGL; o
- Está ejecutando un programa llamante generado por EGL que llama a un programa escrito en EGL.

Las situaciones anteriores pueden producirse fuera del Entorno de prueba de WebSphere, pero también dentro de ese entorno, como cuando un código JSP

invoca un programa escrito con EGL. En cualquier caso, el efecto que se produce es la invocación al código fuente EGL, no a un programa generado por EGL.

Si utiliza el Entorno de prueba de WebSphere, los programas llamador y llamado deben ejecutarse allí; la llamada no puede proceder de una máquina remota.

Si utiliza DEBUG, debe establecer las siguientes propiedades en el mismo elemento **callLink**:

- **library**, que nombra el proyecto que contiene el programa llamado
- **package**, que identifica el paquete que contiene el programa llamado; sin embargo, no es necesario establecer esta propiedad si los programas llamador y llamado está en el mismo paquete

Si el llamador no se ejecuta en el depurador EGL y no se ejecuta en el Entorno de prueba de WebSphere, debe establecer estas propiedades del elemento **callLink**:

- **serverid**, que debe especificar el número de puerto del escuchador si no es el 8346; y
- **location**, que debe contener el nombre de sistema principal de la máquina en la que se ejecuta el entorno de trabajo de Eclipse.

#### **DIRECT**

Especifica que el programa o envoltura llamante utiliza una llamada directa local, lo que significa que el código llamante y el llamado se ejecutan en la misma hebra. No está implicado ningún escuchador TCP/IP y el valor de la propiedad **location** se pasa por alto. **DIRECT** es el valor por omisión.

Un programa Java llamante no utiliza el middleware EGL, pero una envoltura llamante utiliza dicho middleware Java.

Si el código Java generado por EGL está llamando a una biblioteca de enlaces dinámicos (DLL) no generada por EGL o a un programa C o C++, es recomendable que utilice el valor **DISTINCT** de **remoteComType**.

#### **DISTINCT**

Especifica que se inicia una unidad de ejecución nueva al llamar a un programa localmente. La llamada se sigue considerando remota debido a que el middleware EGL está implicado.

Puede utilizar este valor para un programa Java generado por EGL que llama a una biblioteca de enlaces dinámicos (DLL) o a un programa C o C++.

#### **CICSECI**

Especifica la utilización de la interfaz ECI de la Pasarela de transacción CICS (CTG), que es necesaria al depurar o ejecutar código no J2EE que accede a CICS.

Las clases Java CTG se utilizan para implementar este protocolo. Para especificar el URL y el puerto de un servidor CTG, asigne valores al elemento **callLink**, propiedades **ctgLocation** y **ctgPort**. Para identificar la región CICS en la que reside el programa llamado, especifique la propiedad **location**.

#### **CICSJ2C**

Especifica la utilización de un conector J2C para la Pasarela de transacción CICS.

## CICSSSL

Especifica la utilización de características SSL (Capa de sockets segura) de la Pasarela de transacción CICS (CTG). La implementación JSSE de SSL está soportada.

Las clases Java CTG se utilizan para implementar este protocolo. Para especificar información adicional para un servidor CTG, asigne valores a las siguientes propiedades del elemento callLink:

- ctgKeyStore
- ctgKeyStorePassword
- ctgLocation
- ctgPort, que en este caso es el puerto TCP/IP en el que un escuchador CTG JSSE está a la escucha de las peticiones. Si no se especifica ctgPort, se utiliza el puerto por omisión CTG 8050.

Para identificar la región CICS en la que reside el programa llamado, especifique la propiedad location.

## JAVA400

Especifica la utilización de IBM Toolbox para Java para comunicarse entre una envoltura o programa Java y un programa COBOL generado (por EGL o VisualAge Generator) para iSeries.

## TCPIP

Especifica que el middleware EGL utiliza TCP/IP.

## Conceptos relacionados

“Componente de opciones de enlace” en la página 311

## Tareas relacionadas

“Editar el elemento callLink de un componente de opciones de enlace” en la página 314

## Consulta relacionada

“Propiedad ctgKeyStore del elemento callLink” en la página 410

“Propiedad ctgKeyStorePassword del elemento callLink” en la página 410

“Propiedad ctgLocation del elemento callLink” en la página 410

“Propiedad ctgPort del elemento callLink” en la página 411

“Editar el elemento callLink de un componente de opciones de enlace” en la página 314

“Configuración del servidor J2EE para llamadas CICSJ2C” en la página 358

“Config. escucha TCP/IP para aplic. llamada en módulo cliente aplic. J2EE” en la página 359

“Configurar el escucha TCP/IP para una aplicación no J2EE llamada” en la página 353

## Propiedad remotePgmType del elemento callLink

El componente de opciones de enlace, elemento callLink, propiedad **remotePgmType** especifica el tipo de programa al que se llama. La propiedad es aplicable en la siguiente situación:

- El valor de la propiedad **type** es **ejbCall** o **remoteCall**; y
- Se aplica una de las siguientes condiciones:
  - Si está generando un programa (en lugar de una envoltura), la propiedad **pgmName** hace referencia a un programa llamado por el programa que se genera.

El programa llamado es de uno de los siguientes tipos:

- Un programa Java generado por EGL

- Una biblioteca de enlaces dinámicos (DLL) no generada por EGL o un programa C o C++
- Un programa que se ejecuta en CICS y tiene mandatos CICS
- Si está generando una envoltura Java, **pgmName** hace referencia al programa al que se llama por medio de la envoltura Java

#### EGL

El programa llamado es un programa COBOL o Java generado por EGL o por VisualAge Generator; en este caso, el llamador es un , un programa Java o una envoltura Java. Este es el valor por omisión.

#### Conceptos relacionados

“Componente de opciones de enlace” en la página 311  
 “Configuraciones de tiempo de ejecución” en la página 9

#### Tareas relacionadas

“Editar el elemento callLink de un componente de opciones de enlace” en la página 314

#### Consulta relacionada

“Elemento callLink” en la página 407  
 “Propiedad library del elemento callLink” en la página 412  
 “Propiedad pgmName del elemento callLink” en la página 417  
 “Propiedad type del elemento callLink” en la página 423

## Propiedad serverID del elemento callLink

El componente de opciones de enlace, elemento callLink, propiedad **serverID** especifica uno de los siguientes valores:

- El número de puerto TCP/IP del escucha de un programa llamado; pero solamente si se está utilizando el protocolo TCP/IP. En este caso, no existe un valor por omisión.
- El ID de una transacción de CICS a la que se invoca, pero solamente cuando el acceso a CICS lo realizan las características de interfaz ECI o capa de sockets segura de la Pasarela de transacciones de CICS. En este caso, el valor por omisión es la transacción duplicada del sistema servidor CICS.

La propiedad sólo se utiliza en la siguiente situación:

- El valor de la propiedad **type** es **ejbCall** o **remoteCall**;
- El valor de **remoteComType** es **TCPIP**, **CICSECI** o **CICSSSL**; y
- Está generando un programa o envoltura Java
  - Si está generando un programa Java, la propiedad **pgmName** hace referencia a un programa llamado por el programa que se genera.
  - Si está generando una envoltura Java, **pgmName** hace referencia a un programa al que se llama por medio de la envoltura Java.

#### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

#### Tareas relacionadas

“Editar el elemento callLink de un componente de opciones de enlace” en la página 314  
 “Config. escucha TCP/IP para aplic. llamada en módulo cliente aplic. J2EE” en la página 359  
 “Configurar el escucha TCP/IP para una aplicación no J2EE llamada” en la página 353

#### Consulta relacionada

“Elemento callLink” en la página 407

“Propiedad `pgmName` del elemento `callLink`” en la página 417  
“Propiedad `remoteComType` del elemento `callLink`” en la página 419  
“Propiedad `type` del elemento `callLink`”

## Propiedad `type` del elemento `callLink`

El componente de opciones de enlace, elemento `callLink`, propiedad `type` especifica el tipo de llamada. Seleccione uno de los siguientes valores:

### `ejbCall`

Indica que el programa o envoltura Java generado implementará la llamada de programa utilizando un bean de sesión EJB y que éste accederá al programa identificado en la propiedad `pgmName`. El valor `ejbCall` es aplicable en cualquiera de estos dos casos:

- Está generando una envoltura Java y la envoltura llama a ese programa por medio de un bean de sesión EJB. En este caso, la propiedad `pgmName` hace referencia al programa llamado por la envoltura, y la utilización de `ejbCall` provoca la generación del bean de sesión EJB.
- Está generando un programa Java que llama a un programa mediante un bean de sesión EJB. En este caso, la propiedad `pgmName` hace referencia al programa llamado, y no se genera ningún bean de sesión EJB.

En ambos los casos, si utiliza un bean de sesión EJB, debe generar una envoltura Java, aunque sólo sea para generar el bean de sesión EJB.

El bean de sesión generado debe desplegarse en un servidor Java de empresa, y debe cumplirse una de las siguientes condiciones:

- El servidor de nombres utilizado para localizar el bean de sesión EJB debe residir en la misma máquina que el código que llama al bean de sesión; o
- La propiedad `providerURL` debe identificar dónde reside el servidor de nombres.

Si desea utilizar un bean de sesión EJB, debe generar el programa o envoltura llamador con un componente de opciones de enlace en el que el valor de la propiedad `type` del programa llamado sea `ejbCall`. No puede tomar la decisión de utilizar un bean de sesión durante el despliegue. Sin embargo, si establece la propiedad `remoteBind` en `RUNTIME`, puede decidir durante el despliegue cómo accede al bean de sesión EJB al programa generado, aunque es más eficiente tomar de esta decisión durante la generación.

### `localCall`

Especifica que la llamada *no* utiliza el middleware EGL. En este caso, el programa llamado está en el mismo proceso que el llamador.

`localCall` es el valor por omisión

### `remoteCall`

Especifica que la llamada utiliza el middleware EGL, que añade 12 bytes al final de los datos pasados. Dichos bytes permiten al llamador recibir un valor de retorno desde el programa llamado.

Si el llamador es un código Java la comunicación la maneja el protocolo especificado en la propiedad `remoteComType`; la elección de protocolo indica si el programa llamado está en la misma hebra o en otra.

Si en una llamada se pasan registros de longitud variable, se cumple lo siguiente:

- Se reserva espacio para la longitud máxima especificada para un registro

- Si el valor de la propiedad **type** de `callLink` es `remoteCall` o `ejbCall`, el elemento de longitud variable (si existe) debe estar dentro del registro

#### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

#### Tareas relacionadas

“Editar el elemento `callLink` de un componente de opciones de enlace” en la página 314

#### Consulta relacionada

“Elemento `callLink`” en la página 407

“Propiedad `linkType` del elemento `callLink`” en la página 412

“Propiedad `location` del elemento `callLink`” en la página 413

“Propiedad `parmForm` del elemento `callLink`” en la página 416

“Propiedad `pgmName` del elemento `callLink`” en la página 417

“`providerURL` en elemento `callLink`” en la página 417

“Propiedad `remoteComType` del elemento `callLink`” en la página 419

---

## Funciones C con EGL

Los programas EGL pueden invocar funciones C.

#### Para invocar una función C desde EGL:

Después de identificar las funciones C para utilizarlas en el programa EGL, debe:

1. Descargar en el sistema la biblioteca de pila EGL y el archivo de objeto de aplicación del sitio Web de IBM.
2. Compilar todo el código C en una biblioteca compartida y enlazarla con la biblioteca de pila específica de plataforma adecuada.
3. Crear una tabla de función.
4. Compilar la tabla de función y el archivo de objeto de aplicación específico de plataforma adecuado en una biblioteca compartida y enlazar esta biblioteca compartida con la biblioteca compartida creada en el paso 2 y la biblioteca de pila.

#### 1. Descargar la biblioteca de pila EGL y el archivo de objeto de aplicación

Para descargar la biblioteca de pila EGL y el archivo de objeto de aplicación:

1. Vaya al el sitio Web de soporte de EGL.
  - El URL de Rational Application Developer es:  
`http://www3.software.ibm.com/ibmdl/pub/software/rationalsdp/rad/60/redist`
  - El URL de Rational Web Developer es:  
`http://www3.software.ibm.com/ibmdl/pub/software/rationalsdp/rwd/60/redist`
2. Descargue el archivo **EGLRuntimesV60IFix001.zip** en el directorio que prefiera.
3. Desempaque el archivo **EGLRuntimesV60IFix001.zip** para identificar los archivos siguientes:

Para las bibliotecas de pila específicas de la plataforma:

- AIX: `EGLRuntimes/Aix/bin/libstack.so`
- Linux: `EGLRuntimes/Linux/bin/libstack.so`
- Win32:

EGLRuntimes/Win32/bin/stack.dll  
EGLRuntimes/Win32/bin/stack.lib

Para los archivos de objeto de aplicación específicos de la plataforma:

- AIX: EGLRuntimes/Aix/bin/application.o
- Linux: EGLRuntimes/Linux/bin/application.o
- Win32: EGLRuntimes/Win32/bin/application.obj

## 2. Compilar todo el código C en una biblioteca compartida

El código C recibe valores de EGL utilizando funciones externas de recepción y devuelve valores a EGL utilizando funciones externas de devolución. Las funciones externas de extracción se describen en el apartado *Recibir valores de EGL*; las funciones externas de devolución se describen en el apartado *Devolver valores a EGL*.

Para compilar todo el código C en una biblioteca compartida:

1. Utilizando métodos estándar, compile todo el código C en una biblioteca compartida y enlázelo con la biblioteca de pila EGL específica de plataforma adecuada.
2. En los ejemplos específicos de plataforma siguientes, **file1.c** y **file2.c** son archivos C que contienen funciones invocadas por EGL.

En AIX (el mandato ld debe estar en una sola línea):

```
cc -c -Iincl_dir file1.c file2.c
ld -G -b32 -bexpall -bnoentry
 -brtl file1.o file2.o -Ldir_bib_pila
 -lstack -o nombre_lib1 -lc
```

En Linux (el mandato gcc debe estar en una sola línea):

```
cc -c -Iincl_dir file1.c file2.c
gcc -shared file1.o file2.o -Ldir_bib_pila
 -lstack -o nombre_lib1
```

En Windows (el mandato de enlace debe estar en una sola línea):

```
cl /c -Iincl_dir file1.c file2.c
link /DLL file1.obj file2.obj
 /LIBPATH:dir_bib_pila
 /DEFAULTLIB:stack.lib /OUT:nombre_bib1
```

*incl\_dir*

la ubicación del directorio de los archivos de cabecera.

*dir\_bib\_pila*

La ubicación del directorio de la biblioteca de pila.

*nombre\_bib1*

el nombre de la biblioteca de salida.

**Nota:** Si el código C utiliza cualquiera de las funciones de biblioteca ESQL/C de IBM Informix (BIGINT, DECIMAL, DATE, INTERVAL, DATETIME), entonces también debe enlazarse la biblioteca ESQL/C.

## 3. Crear una tabla de función

La tabla de funciones es un archivo fuente C que incluye los nombres de todas las funciones C que debe invocar el programa EGL. En la tabla de funciones siguiente, por ejemplo, `c_fun1` y `c_fun2` son nombres de funciones C. Todas las funciones identificadas en el código deben haberse exportado desde la biblioteca C creada en el paso 2 anterior.

```
#include <stdio.h>
struct func_table {

 char *fun_name;
 int (*fptr)(int);
};

extern int c_fun1(int);
extern int c_fun2(int);
/* Prototipos parecidos para otras funciones */

struct func_table ftab[] =
{
 "c_fun1", c_fun1,
 "c_fun2", c_fun2,
 /* Igualmente para otras funciones */
 "", NULL
};
```

Cree una tabla de funciones basada en el ejemplo anterior y púBLELA con las funciones C adecuadas. Indique el final de la tabla de funciones con "", NULL.

#### 4. Compilar la tabla de funciones y el archivo de objeto de aplicación específico de plataforma en una biblioteca compartida

El archivo de objeto de aplicación es la interfaz entre el código EGL y el código C.

Los dos artefactos siguientes deben compilarse en una biblioteca compartida y enlazarse con la biblioteca de pila y la biblioteca creada en el paso 2 anterior:

- tabla de función
- archivo de objeto de aplicación

Compile la biblioteca compartida nueva utilizando el ejemplo siguiente, donde **ftable.c** es el nombre de la tabla de función y **mylib** es el nombre de la biblioteca compartida C creada en el paso 2 y **lib\_dir** es la ubicación para **mylib**. Especifique **lib2\_name** utilizando la propiedad *dllName* o la propiedad de tiempo de ejecución Java *vgj.defaultI4GLNativeLibrary*.

En AIX (el mandato `ld` debe estar en una sola línea):

```
cc -c ftable.c
ld -G -b32 -bexpall -bnoentry
 -brtl ftable.o application.o
 -Lstack_lib_dir -lstack -llib_dir
 -lmylib -o lib2_name -lc
```

En Linux (el mandato `gcc` debe estar en una sola línea):

```
cc -c ftable.c
gcc -shared ftable.o application.o
 -Lstack_lib_dir -lstack -llib_dir
 -lmylib -o lib2_name
```

En Windows (el mandato `link` debe estar en una sola línea):

```

cl /c ftable.c
link /DLL ftable.obj application.obj
 /LIBPATH:stack_lib_dir
 /DEFAULTLIB:stack.lib
 /LIBPATH:lib_dir
 /DEFAULTLIB:mylib.lib /OUT:lib2_name

```

Enlace las tres bibliotecas entre sí.

Con la biblioteca compartida C, la tabla de función y la biblioteca de pila enlazadas, ahora puede invocar las funciones C desde el código EGL. Para obtener información acerca de cómo invocar una función C en EGL, consulte el apartado *Invocar una función C desde un programa EGL*.

### Concepto relacionado

“Componente de opciones de enlace” en la página 311

### Consulta relacionada

“Funciones BIGINT para C”

“Tipos de datos C y tipos primitivos EGL” en la página 428

“Funciones DATE para C” en la página 429

“Funciones DATETIME e INTERVAL para C” en la página 430

“Funciones DECIMAL para C” en la página 431

“Invocar una función C desde un programa EGL” en la página 432

“Funciones de devolución para C” en la página 437

“Funciones de pila para C” en la página 434

## Funciones BIGINT para C

**Nota:** La funcionalidad BIGINT siguiente solo está disponible para los usuarios de IBM Informix ESQL/C. Para utilizar estas funciones, los usuarios de ESQL/C necesitarán enlazar manualmente el código C con las bibliotecas ESQL/C.

El tipo de datos BIGINT es un método independiente del sistema para representar números en el rango de  $-2^{63}-1$  a  $2^{63}-1$ . ESQL/C proporciona rutinas que facilitan la conversión del tipo de datos BIGINT a otros tipos de datos del lenguaje C.

El tipo de datos BIGINT está representado internamente con la estructura **ifx\_int8\_t**. Encontrará información acerca de la estructura en el archivo de cabecera **int8.h** que se encuentra en el producto ESQL/C. Incluya este archivo en todos los archivos fuente C que utilicen cualquiera de las funciones BIGINT.

Todas las operaciones con números de tipo **int8** deben realizarse utilizando las funciones de biblioteca ESQL/C siguientes para el tipo de datos **int8**. Cualesquiera otras operaciones, modificaciones o análisis pueden producir resultados impredecibles. La biblioteca ESQL/C proporciona las funciones siguientes que permiten manipular números **int8** y convertir números de tipo **int8** a otros tipos de datos y viceversa.

Nombre de función	Descripción
ifx_int8add( )	Añade dos valores de tipo BIGINT
ifx_int8cmp( )	Compara dos números de tipo BIGINT
ifx_int8copy( )	Copia una estructura <b>ifx_int8_t</b>

Nombre de función	Descripción
ifx_int8cvasc( )	Convierte un valor de tipo C <b>char</b> en un número de tipo BIGINT
ifx_int8cvdbl( )	Convierte un número de tipo C <b>double</b> en un número de tipo BIGINT
ifx_int8cvdec( )	Convierte un valor de tipo <b>decimal</b> en un valor de tipo BIGINT
ifx_int8cvflt( )	Convierte un valor de tipo C <b>float</b> en un valor de tipo BIGINT
ifx_int8cvint( )	Convierte un número de tipo C <b>int</b> en un número de tipo BIGINT
ifx_int8cvlong( )	Convierte un valor de tipo C <b>long</b> ( <b>int</b> en máquinas de 64 bits) en un valor de tipo BIGINT
ifx_int8cvlong_long( )	Convierte un tipo C <b>long long</b> (valor de 8 bytes, <b>long long</b> en 32 bits y <b>long</b> en 64 bits) en un valor de tipo BIGINT
ifx_int8div( )	Divide dos números BIGINT
ifx_int8mul( )	Multiplica dos números BIGINT
ifx_int8sub( )	Resta dos números BIGINT
ifx_int8toasc( )	Convierte un valor de tipo BIGINT en un valor de tipo C <b>char</b>
ifx_int8todbl( )	Convierte un valor de tipo BIGINT en un valor de tipo C <b>double</b>
ifx_int8todec( )	Convierte un número de tipo BIGINT en un número de tipo <b>decimal</b>
ifx_int8toflt( )	Convierte un número de tipo BIGINT en un número de tipo C <b>float</b>
ifx_int8toint( )	Convierte un valor de tipo BIGINT en un valor de tipo C <b>int</b>
ifx_int8tolong( )	Convierte un valor de tipo BIGINT en un valor de tipo C <b>long</b> ( <b>int</b> en una máquina de 64 bits)
ifx_int8tolong_long( )	Convierte un tipo C <b>long long</b> ( <b>long</b> en una máquina de 64 bits) en un valor de tipo BIGINT

### Consulta relacionada

Para obtener más información acerca de las funciones individuales, consulte el documento siguiente: IBM Informix ESQL/C Programmer's Manual.

“Funciones DATE para C” en la página 429

“Funciones DATETIME e INTERVAL para C” en la página 430

“Funciones DECIMAL para C” en la página 431

“Invocar una función C desde un programa EGL” en la página 432

## Tipos de datos C y tipos primitivos EGL

La tabla siguiente muestra la correlación entre tipos de datos C, tipos de datos I4GL y tipos primitivos EGL.

Tipos de datos C	Tipo de datos I4GL equivalente	Tipo primitivo EGL equivalente
char	CHAR o CHARACTER	UNICODE(1)
char	NCHAR	UNICODE(tamaño)

Tipos de datos C	Tipo de datos I4GL equivalente	Tipo primitivo EGL equivalente
char	NVARCHAR	STRING
char	VARCHAR	STRING
int	INT o INTEGER	INT
short	SMALLINT	SMALLINT
ifx_int8_t	BIGINT	BIGINT
dec_t	DEC o DECIMAL(p,s,) o NUMERIC(p)	DECIMAL(p)
dec_t	MONEY	MONEY
double	FLOAT	FLOAT
float	SMALLFLOAT	SMALLFLOAT
loc_t	TEXT	CLOB
loc_t	BYTE	BLOB
int	DATE	DATE
dtime_t	DATETIME	TIMESTAMP
intvl_t	INTERVAL	INTERVAL

#### Consulta relacionada

“BIN y los tipos enteros (integer)” en la página 50

“BLOB” en la página 49

“CLOB” en la página 48

“DATE” en la página 41

“DECIMAL” en la página 50

“FLOAT” en la página 51

“INTERVAL” en la página 42

“Invocar una función C desde un programa EGL” en la página 432

“MBCHAR” en la página 39

“MONEY” en la página 51

“NUM” en la página 51

“Tipos primitivos” en la página 34

“SMALLFLOAT” en la página 53

“TIME” en la página 43

“TIMESTAMP” en la página 44

## Funciones DATE para C

**Nota:** La funcionalidad DATE siguiente solo está disponible para los usuarios de IBM Informix ESQL/C. Para utilizar estas funciones, los usuarios de ESQL/C necesitarán enlazar manualmente el código C con las bibliotecas ESQL/C.

Las funciones de manipulación de datos siguientes están en la biblioteca ESQL/C. Convierten fechas entre un formato de serie y el formato DATE interno.

Nombre de función	Descripción
rdatestr( )	Convierte un formato DATE interno en un formato de serie de caracteres

Nombre de función	Descripción
rdayofweek( )	Devuelve el día de la semana de una fecha en formato interno
rdefmtdate( )	Convierte un formato de serie especificado en un formato DATE interno
rfmtdate( )	Convierte un DATE interno en un formato de serie especificado
rjulmdy( )	Devuelve mes, día y año a partir un formato DATE especificado
rleapyear( )	Determina si el año especificado es un año bisiesto
rmdyjul( )	Devuelve un formato DATE interno a partir de un mes, un día y un año
rstrdate( )	Convierte un formato de serie de caracteres en un formato DATE interno
rtoday( )	Devuelve una fecha del sistema como un formato DATE interno

### Consulta relacionada

Para obtener más información acerca de las funciones individuales, consulte el documento siguiente: IBM Informix ESQL/C Programmer's Manual.

"Funciones BIGINT para C" en la página 427

"Funciones DATETIME e INTERVAL para C"

"Funciones DECIMAL para C" en la página 431

"Invocar una función C desde un programa EGL" en la página 432

## Funciones DATETIME e INTERVAL para C

**Nota:** La funcionalidad DATETIME e INTERVAL siguiente solo está disponible para los usuarios de IBM Informix ESQL/C. Para utilizar estas funciones, los usuarios de ESQL/C necesitarán enlazar manualmente el código C con las bibliotecas ESQL/C.

Los tipos de datos DATETIME e INTERVAL se representan internamente con las estructuras **dtime\_t** e **intrvl\_t** respectivamente. Encontrará información acerca de estas estructuras en el archivo de cabecera **datetime.h** que se encuentra en el producto ESQL/C. Incluya este archivo en todos los archivos fuente C que utilicen cualquiera de las funciones DATETIME e INTERVAL.

Debe utilizar las funciones de biblioteca ESQL/C siguientes para los tipos de datos **datetime** e **interval** para realizar operaciones sobre estos tipos de datos.

Nombre de función	Descripción
dtaddinv( )	Añade un valore de intervalo a un valor de fecha y hora
dtcurrent( )	Obtiene la fecha y la hora actuales
dtcvasc( )	Convierte una serie de caracteres ANSI en un valor de fecha y hora
dtcvfntasc( )	Convierte una serie de caracteres con un formato especificado en un valor de fecha y hora

Nombre de función	Descripción
dtextend( )	Cambia el calificador de un valor de fecha y hora
dtsub( )	Resta un valor de fecha y hora de otro
dsubinv()	Resta un valor de intervalo de un valor de fecha y hora
dttoasc( )	Convierte un valor de fecha y hora en una serie de caracteres ANSI
dttofmtasc( )	Convierte un valor de fecha y hora en una serie de caracteres con un formato especificado
incvasc( )	Convierte una serie de caracteres ANSI en un valor de intervalo
incvfmtasc( )	Convierte una serie de caracteres con un formato especificado en un valor de intervalo
intoasc( )	Convierte un valor de intervalo en una serie de caracteres ANSI
intofmtasc( )	Convierte un valor de intervalo en una serie de caracteres con un formato especificado
invdivdbl( )	Divide un valor de intervalo por un valor numérico
invdivinv( )	Divide un valor de intervalo por otro valor de intervalo
invextend( )	Amplía un valor de intervalo a un calificador de intervalo distinto
invmuldbl( )	Multiplifica un valor de intervalo por un valor numérico

### Consulta relacionada

Para obtener más información acerca de las funciones individuales, consulte el documento si IBM Informix ESQL/C Programmer's Manual.

“Funciones BIGINT para C” en la página 427

“Funciones DATE para C” en la página 429

“Funciones DECIMAL para C”

“Invocar una función C desde un programa EGL” en la página 432

## Funciones DECIMAL para C

**Nota:** La funcionalidad DECIMAL siguiente solo está disponible para los usuarios de IBM Informix ESQL/C. Para utilizar estas funciones, los usuarios de ESQL/C necesitarán enlazar manualmente el código C con las bibliotecas ESQL/C.

El tipo de datos DECIMAL es un método independiente del sistema para la representación de hasta 32 dígitos significativos, con o sin una coma decimal y con exponentes en el rango de -128 a +126. ESQL/C proporciona rutinas que facilitan la conversión de números de tipo DECIMAL a y desde cada tipo de datos permitido en el lenguaje C. Los números de tipo DECIMAL constan de un exponente y una mantisa (o componente fraccional) en base 100. En formato normalizado, el primer dígito de la mantisa debe ser mayor que cero.

El tipo de datos DECIMAL se representa internamente con la estructura **dec\_t**. La estructura **decimal** y la definición de tipo **dec\_t** pueden encontrarse en el archivo

de cabecera **decimal.h**, que está incluido en el producto ESQL/C. Incluya este archivo en todos los archivos fuente C que utilicen cualquiera de las funciones decimales.

Todas las operaciones con números de tipo **decimal** deben realizarse utilizando las funciones de biblioteca ESQL/C siguientes para el tipo de datos **decimal**. Cualesquiera otras operaciones, modificaciones o análisis pueden producir resultados impredecibles.

Nombre de función	Descripción
deccvasc( )	Convierte el tipo C <b>int1</b> en el tipo DECIMAL
dectoasc( )	Convierte el tipo DECIMAL en el tipo C <b>int1</b>
deccvint( )	Convierte el tipo C <b>int</b> en el tipo DECIMAL
dectoint( )	Convierte el tipo DECIMAL en el tipo C <b>int</b>
deccvlong( )	Convierte el tipo C <b>int4</b> en el tipo DECIMAL
dectolong( )	Convierte el tipo DECIMAL en el tipo C <b>int4</b>
deccvflt( )	Convierte el tipo C <b>float</b> en el tipo DECIMAL
dectoflt( )	Convierte el tipo DECIMAL en el tipo C <b>float</b>
deccvdbl( )	Convierte el tipo C <b>double</b> en el tipo DECIMAL
dectodbl( )	Convierte el tipo DECIMAL en el tipo C <b>double</b>
decadd( )	Añade dos números DECIMAL
decsub( )	Resta dos números DECIMAL
decmul( )	Multiplica dos números DECIMAL
decdiv( )	Divide dos números DECIMAL
deccmp( )	Compara dos números DECIMAL
deccopy( )	Copia un número DECIMAL
dececv( )	Convierte un valor DECIMAL en una serie ASCII
decfcvt( )	Convierte un valor DECIMAL en una serie ASCII

### Consulta relacionada

Para obtener más información acerca de las funciones individuales, consulte el documento siguiente: IBM Informix ESQL/C Programmer's Manual.

"Funciones BIGINT para C" en la página 427

"Funciones DATE para C" en la página 429

"Funciones DATETIME e INTERVAL para C" en la página 430

"Invocar una función C desde un programa EGL"

## Invocar una función C desde un programa EGL

Puede invocar (o llamar) una función C desde un programa EGL. Antes de seguir las instrucciones que se proporcionan a continuación, debe compilar y enlazar el código C tal como se indica en el apartado *Funciones C con EGL*.

Para invocar una función C desde un programa EGL:

1. Utilizando la sentencia *invocación de función*, especifique lo siguiente:
  - El nombre de la función C
  - Argumentos que hay que pasar a la función C
  - Variables a devolver al programa EGL

2. Cree un *componente de biblioteca* EGL nativo que contenga la definición de la función.
3. Con la sentencia USE, especifique el componente nativo de EGL en el módulo de llamada.

Por ejemplo, la sentencia de invocación de función siguiente llama a la función C **sendmsg()**

```
sendmsg(chartype, 4, msg_status, return_code);
```

Pasa dos argumentos (**chartype** y **4**, respectivamente) a la función y espera que se devuelvan dos argumentos (**msg\_status** y **return\_code**, respectivamente). Esto queda claro al definir la función en una biblioteca nativa de la forma siguiente:

```
Library I4GLFunctions type nativeLibrary
 {callingConvention = "I4GL", dllName = "mydll"}
 Function sendmsg(chartype char(10) in, i int in, msg_status int out, return_code int out)
 end
end
```

Los argumentos que se pasan se especifican utilizando el parámetro "in" y los argumentos a devolver se especifican utilizando el parámetro "out".

*convenioLlamada*

especifica que los argumentos se pasarán entre funciones y el código de llamada utilizando el mecanismo de pila de argumentos.

*nombreDll*

Especifica la biblioteca compartida de C en la que existe esta función.

**Nota:** El nombre de la biblioteca compartida de C también puede especificarse utilizando la propiedad del sistema *vgj.defaultI4GLNativeLibrary*. Si se han especificado tanto *nombreDll* como la propiedad del sistema, se utilizará *nombreDll*. Para obtener más información acerca de la biblioteca nativa de EGL, consulte el tema de ayuda *Componente de biblioteca de tipo nativeLibrary*.

La función C recibe un argumento entero que especifica cuántos valores se han puesto en la pila de argumentos (en este caso, dos argumentos). Este es el número de valores que se deben sacar de la pila en la función C. La función también necesita devolver valores para los argumentos **msg\_status** y **return\_code** antes de devolver el control al programa EGL. Las funciones externas de extracción se describen en el apartado *Recibir valores de EGL*; las funciones externas de devolución se describen en el apartado *Devolver valores a EGL*.

La función C no debe dar por supuesto que se haya pasado el número correcto de valores apilados. La función C debe probar el argumento entero para ver cuántos argumentos EGL tiene apilados.

En este ejemplo se muestra una función C que necesita exactamente un argumento:

```
int nxt_bus_day(int nargs);
{
 int theDate;
 if (nargs != 1)
 {
 fprintf(stderr,
 "nxt_bus_day: wrong number of parms (%d)\n",
 nargs);
 ibm_lib4gl_returnDate(0L);
 return(1);
 }
 ibm_lib4gl_popDate(&theDate);
```

```

switch(rdayofweek(theDate))
{
case 5: /* change friday -> monday */
 ++theDate;
case 6: /* saturday -> monday*/
 ++theDate;
default: /* (sun..thur) go to next day */
 ++theDate;
}
ibm_lib4gl_returnDate(theDate); /* stack result */
return(1) /* return count of stacked */
}

```

La función devuelve la fecha del siguiente día hábil después de una fecha dada. Puesto que la función debe recibir exactamente un argumento, la función comprueba el número de argumentos pasados. Si la función recupera un número distinto de argumentos, termina el programa (con un mensaje de identificación.)

#### Consulta relacionada

“Funciones BIGINT para C” en la página 427  
 “Tipos de datos C y tipos primitivos EGL” en la página 428  
 “Crear un componente de biblioteca de EGL” en la página 141  
 “Funciones DATE para C” en la página 429  
 “Funciones DATETIME e INTERVAL para C” en la página 430  
 “Funciones DECIMAL para C” en la página 431  
 “Invocaciones de función” en la página 518  
 “Componente de biblioteca de tipo basicLibrary” en la página 142  
 “Funciones de pila para C”  
 “Funciones de devolución para C” en la página 437  
 “Funciones C con EGL” en la página 424

## Funciones de pila para C

Para llamar una función C, EGL utiliza una *pila de argumentos*, un mecanismo que pasa argumentos entre las funciones y el código de llamada. La función de llamada de EGL pone los argumentos en la pila y la función C llamada los saca de la pila para utilizar los valores. La función llamada pone los valores de retorno en la pila y el llamador los saca para recuperar los valores. Las funciones externas de extracción y de llamada se proporcionan con la biblioteca de pila de argumentos. Las funciones externas de extracción se describen a continuación, según el tipo de datos del valor que cada una extrae de la pila de argumentos. Las funciones externas de devolución se describen en el apartado *Funciones de devolución para C*.

**Nota:** Las funciones de extracción se utilizaban originalmente con IBM Informix 4GL (I4GL), de aquí la inclusión de “4gl” en los nombres de función.

#### Funciones de biblioteca para devolver valores

Puede llamar a las funciones de biblioteca siguientes desde una función C para extraer valores de número de la pila de argumentos:

- extern void ibm\_lib4gl\_popMInt(int \*iv)
- extern void ibm\_lib4gl\_popInt2(short \*siv)
- extern void ibm\_lib4gl\_popInt4(int \*liv)
- extern void ibm\_lib4gl\_popFloat(float \*fv)
- extern void ibm\_lib4gl\_popDouble(double \*dfv)
- extern void ibm\_lib4gl\_popDecimal(dec\_t \*decv)
- extern void ibm\_lib4gl\_popInt8(ifx\_int8\_t \*bi)

Las tablas siguientes correlacionan los nombres de función de retorno entre I4GL anterior a la Versión 7.31 y la Versión 7.31 y posteriores:

Nombre anterior a la Versión 7.31	Nombre de la Versión 7.31 y posteriores
popint	ibm_lib4gl_popMInt
popshort	ibm_lib4gl_popInt2
poplong	ibm_lib4gl_popInt4
popflo	ibm_lib4gl_popFloat
popdub	ibm_lib4gl_popDouble
popdec	ibm_lib4gl_popDecimal

Cada una de estas funciones, como todas las funciones de biblioteca para extraer valores, realiza las acciones siguientes:

1. Elimina un valor de la pila de argumentos.
2. Convierte el tipo de datos si es necesario. Si el valor de la pila no puede convertirse al tipo especificado, se produce un error.
3. Copia el valor en la variable designada.

Los tipos de estructura **dec\_t** y **ifx\_int8\_t** se utilizan para representar datos DECIMAL y BIGINT en un programa C. Para obtener más información acerca de los tipos de estructura **dec\_t** y **ifx\_int8\_t** y de las funciones de biblioteca para manipular e imprimir las variables DECIMAL y BIGINT, consulte la publicación *IBM Informix ESQL/C Programmer's Manual*.

### Funciones de biblioteca para extraer series de caracteres

Puede llamar las funciones de biblioteca siguientes para extraer valores de caracteres:

- extern void ibm\_lib4gl\_popQuotedStr(char \*qv, int len)
- extern void ibm\_lib4gl\_popString(char \*qv, int len)
- extern void ibm\_lib4gl\_popVarChar(char \*qv, int len)

Nombre anterior a la Versión 7.31	Nombre de la Versión 7.31 y posteriores
popquote	ibm_lib4gl_popQuotedStr
popstring	ibm_lib4gl_popString
popvchar	ibm_lib4gl_popVarChar

Tanto **ibm\_lib4gl\_popQuotedStr( )** como **ibm\_lib4gl\_popVarChar( )** copian exactamente **len** bytes en el almacenamiento intermedio de series **\*qv**. Aquí **ibm\_lib4gl\_popQuotedStr( )** se rellena con espacios según sea necesario, pero **ibm\_lib4gl\_popVarChar( )** no se rellena en toda su longitud. El byte final copiado en el almacenamiento intermedio es un byte nulo para terminar la serie por lo que la longitud de datos de serie máxima es **len-1**. Si el argumento de pila es mayor que **len-1**, se pierden los bytes de cola.

El argumento **len** establece el tamaño máximo del almacenamiento de serie de recepción. Utilizando **ibm\_lib4gl\_popQuotedStr( )**, se reciben exactamente **len** bytes (incluyendo espacios en blanco de cola y el nulo), incluso si el valor de la

pila es una serie vacía. Para averiguar la longitud de datos verdadera de una serie recuperada por `ibm_lib4gl_popQuotedStr( )`, debe recortar los espacios de cola del valor extraído.

**Nota:** Las funciones `ibm_lib4gl_popString( )` y `ibm_lib4gl_popQuotedStr( )` son idénticas, excepto por el hecho de que `ibm_lib4gl_popString( )` recorta automáticamente los blancos de cola.

### Funciones de biblioteca para extraer valores de hora

Puede llamar las funciones de biblioteca siguientes para extraer valores DATE, INTERVAL y DATETIME (TIMESTAMP):

- `extern void ibm_lib4gl_popDate(int *datv)`
- `extern void ibm_lib4gl_popInterval(intrvl_t *iv, int qual)`

Puede llamar la función de biblioteca siguiente para extraer valores TIMESTAMP:

- `extern void ibm_lib4gl_popDateTime(dtime_t *dtv, int qual)`

Nombre anterior a la Versión 7.31	Nombre de la Versión 7.31 y posteriores
popdate	ibm_lib4gl_popDate
popdtime	ibm_lib4gl_popDateTime
popinv	ibm_lib4gl_popInterval

Los tipos de estructura `dtime_t` y `intrvl_t` se utilizan para representar datos DATETIME e INTERVAL en un programa C. El argumento `qual` recibe la representación binaria del calificador DATETIME o INTERVAL. Para obtener más información acerca de los tipos de estructura `dtime_t` e `intrvl_t` y de las funciones de biblioteca para manipular e imprimir variables DATE, DATETIME e INTERVAL, consulte la publicación *IBM Informix ESQL/C Programmer's Manual*.

### Funciones de biblioteca para extraer valores BYTE o TEXT

Puede llamar la función siguiente para extraer un argumento BYTE o TEXT:

- `extern void ibm_lib4gl_popBlobLocator(loc_t **blob)`

Nombre anterior a la Versión 7.31	Nombre de la Versión 7.31 y posteriores
poplocator	ibm_lib4gl_popBlobLocator

El tipo de estructura `loc_t` define un valor BYTE o TEXT y se trata en la publicación *IBM Informix ESQL/C Programmer's Manual*.

Cualquier argumento BYTE o TEXT debe extraerse como BYTE o TEXT porque EGL no proporciona conversión de tipo de datos automática.

### Consulta relacionada

- “Funciones BIGINT para C” en la página 427
- “Tipos de datos C y tipos primitivos EGL” en la página 428
- “Funciones C con EGL” en la página 424
- “Funciones DATE para C” en la página 429
- “Funciones DATETIME e INTERVAL para C” en la página 430
- “Funciones DECIMAL para C” en la página 431
- “Invocar una función C desde un programa EGL” en la página 432

## Funciones de devolución para C

Para llamar una función C, EGL utiliza una *pila de argumentos*, un mecanismo que pasa argumentos entre las funciones y el código de llamada. La función de llamada de EGL pone los argumentos en la pila y la función C llamada los saca de la pila para utilizar los valores. La función llamada pone los valores de retorno en la pila y el llamador los saca para recuperar los valores. Las funciones externas de extracción y de llamada se proporcionan con la biblioteca de pila de argumentos. Las funciones externas de devolución se describen a continuación; las funciones externas de extracción utilizadas se describen en el apartado *Funciones de pila para C*.

Las funciones de devolución externas copian sus argumentos en el almacenamiento asignado fuera de la función de llamada. Este almacenamiento se libera cuando se extrae el valor devuelto. Esta situación posibilita la devolución de valores de variables locales de la función.

**Nota:** Las funciones de devolución se utilizaban originalmente con IBM Informix 4GL (I4GL), de aquí la inclusión de "4gl" en los nombres de función.

### Funciones de biblioteca para devolver valores

Las funciones de biblioteca siguientes están disponibles para devolver valores:

- extern void ibm\_lib4gl\_returnMInt(int iv)
- extern void ibm\_lib4gl\_returnInt2(short siv)
- extern void ibm\_lib4gl\_returnInt4(int lv)
- extern void ibm\_lib4gl\_returnFloat(float \*fv)
- extern void ibm\_lib4gl\_returnDouble(double \*dfv)
- extern void ibm\_lib4gl\_returnDecimal(dec\_t \*decv)
- extern void ibm\_lib4gl\_returnQuotedStr(char \*str0)
- extern void ibm\_lib4gl\_returnString(char \*str0)
- extern void ibm\_lib4gl\_returnVarChar(char \*vc)
- extern void ibm\_lib4gl\_returnDate(int date)
- extern void ibm\_lib4gl\_returnDateTime(dtime\_t \*dtv)
- extern void ibm\_lib4gl\_returnInterval(intrvl\_t \*inv)
- extern void ibm\_lib4gl\_returnInt8(ifx\_int8\_t \*bi)

La tabla siguiente correlaciona los nombres de función de devolución entre I4GL anteriores a la Versión 7.31 y la Versión 7.31 y posteriores:

Nombre anterior a la Versión 7.31	Nombre de la Versión 7.31 y posteriores
retint	ibm_lib4gl_returnMInt
retshort	ibm_lib4gl_returnInt2
retlong	ibm_lib4gl_returnInt4
retflo	ibm_lib4gl_returnFloat
retdub	ibm_lib4gl_returnDouble
retdec	ibm_lib4gl_returnDecimal
retquote	ibm_lib4gl_returnQuotedStr

Nombre anterior a la Versión 7.31	Nombre de la Versión 7.31 y posteriores
retstring	ibm_lib4gl_returnString
retvchar	ibm_lib4gl_returnVarChar
retdate	ibm_lib4gl_returnDate
retmtime	ibm_lib4gl_returnDateTime
retinv	ibm_lib4gl_returnInterval

El argumento de `ibm_lib4gl_returnQuotedStr( )` es una serie terminada en nulo. La función `ibm_lib4gl_returnString( )` se incluye solo por simetría; internamente llama a `ibm_lib4gl_returnQuotedStr( )`.

La función C puede devolver datos de cualquier forma que sea conveniente. Si la conversión es posible, EGL convierte el tipo de datos según sea necesario al extraer el valor. Si la conversión de tipo de datos no es posible, se produce un error.

Las funciones C llamadas desde EGL deben salir siempre con la sentencia `return(n)`, donde *n* es el número de valores de retorno puestos en la pila. Una función que no devuelve nada debe salir con `return(0)`.

#### Consulta relacionada

- “Funciones BIGINT para C” en la página 427
- “Tipos de datos C y tipos primitivos EGL” en la página 428
- “Invocar una función C desde un programa EGL” en la página 432
- “Funciones C con EGL” en la página 424
- “Funciones DATE para C” en la página 429
- “Funciones DATETIME e INTERVAL para C” en la página 430
- “Funciones DECIMAL para C” en la página 431
- “Funciones de pila para C” en la página 434

---

## Comentarios

Un *comentario* es un archivo EGL creado mediante cualquiera de los siguientes métodos:

- Las barras inclinadas dobles (//) indican que los caracteres subsiguientes constituyen un comentario, incluido el carácter de fin de línea
- Un comentario de una sola o de varias líneas está delimitado por una barra inclinada y un asterisco iniciales (/\*) y por un asterisco y una barra inclinada finales (\*/); este formato de comentario es válido en cualquier lugar en el que lo sea un carácter de espacio en blanco

Puede colocar un comentario dentro o fuera de una sentencia ejecutable, como en este ejemplo:

```
/* la asignación e = f se produce si a == b or if c == d */
if (a == b // una comparación
 || /* OR; otra comparación */ c == d)
 e = f;
end
```

EGL no da soporte a comentarios intercalados, por lo que las entradas siguientes provocarán un error:

```
/* esta línea inicia un comentario */ y
 está línea finaliza el comentario, */
 pero esta línea no está dentro de un comentario */
```

El comentario de las dos primeras líneas incluye un segundo delimitador de comentario (/). Sólo se produce un error cuando EGL intenta interpretar la tercera línea como código fuente.

La siguiente especificación es válida:

```
a = b; /* esta línea inicia un comentario // y
 esta línea finaliza el comentario */
```

Las barras inclinadas dobles (//) del último ejemplo forman parte de un comentario más largo.

Entre los símbolos #sql{ y }, los comentarios de EGL descritos anteriormente no son válidos. Se aplican las siguientes normas:

- Una sentencia SQL se inicia con un guión doble (--) al principio de una línea o después de un espacio en blanco y continúa hasta el final de la línea
- Los comentarios no están disponibles dentro de un literal de serie. Una serie de caracteres en ese literal se interpreta como texto incluso en estos contextos:
  - Una sentencia prepare
  - La propiedad **defaultSelectCondition** de un registro de tipo SQLRecord

#### Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13

#### Consulta relacionada

“Formato fuente EGL” en la página 491

“Sentencias EGL” en la página 88

---

## Compatibilidad con VisualAge Generator

EGL sustituye a VisualAge Generator 4.5 e incluye una sintaxis enfocada principalmente a permitir la migración de los programas existentes al nuevo entorno de desarrollo. La sintaxis está soportada en el entorno de desarrollo si se selecciona la preferencia de EGL **VAGCompatibility** o si (durante la generación o depuración) la opción del descriptor de construcción **VAGCompatibility** se establece en *sí*. La configuración de la preferencia también establece el valor por omisión de la opción del descriptor de construcción.

Se aplica lo siguiente cuando está en vigor la compatibilidad con VisualAge Generator:

- Tres caracteres que no son válidos en otras circunstancias (- @ #) son válidos en los identificadores, aunque el guión (-) y el signo # no son válidos como carácter inicial en ningún caso; para obtener información detallada, consulte la sección *Convenios de denominación*
- Si se hace referencia a una matriz unidimensional estática sin especificar un índice, el índice de la matriz toma por omisión el valor 1; para obtener información detallada, consulte la sección *Matrices*
- Están disponibles los tipos primitivos NUMC y PACF, como se describe en la sección *Tipos primitivos*
- Si se especifica una longitud par para un elemento de tipo primitivo DECIMAL, EGL incrementa la longitud en uno excepto cuando el elemento se utiliza como una variable del lenguaje principal SQL.
- La propiedad de elemento SQL **SQLDataCode** está disponible, como se describe en la sección *Propiedades de elemento SQL*

- Un conjunto de opciones call están disponibles en la sentencia call
- La opción **externallyDefined** está en las sentencias show y transfer
- Las siguientes variables de sistema están disponibles:
  - VGVar.handleSysLibraryErrors
  - ConverseVar.segmentedMode
- Las siguientes funciones de sistema están disponibles:
  - VGLib.getVAGSysType
  - VGLib.connectionService
- Puede emitir una sentencia del siguiente formulario:
 

```
display formularioImpresión
```

*formularioImpresión*

Nombre de un formulario de impresión que está visible en el programa.

En este caso, **display** es equivalente a print.
- Las propiedades de programa siguientes están disponibles en todos los casos y resultan especialmente útiles para el código escrito en VisualAge Generator:
  - **allowUnqualifiedItemReferences**
  - **handleHardIOErrors** (cuando se establece en *no*)
  - **includeReferencedFunctions**
  - **localSQLScope** (cuando se establece en *yes*)
  - **throwNrfEofExceptions** (cuando se establece en *yes*)

Para obtener información detallada, consulte la sección *Componente de programa en formato fuente EGL*.
- Si se establece la propiedad de formulario de texto **value**, el contenido de dicha propiedad sólo está disponible en el programa después de que el usuario haya devuelto el formulario. Por este motivo, no es necesario que el valor que se establece en el programa sea válido para el elemento del programa.

Para acceder a detalles completos sobre la migración de programas de VisualAge Generator a EGL, consulte la sección *Fuentes de información adicional en EGL*.

### Conceptos relacionados

“Fuentes de información adicional acerca de EGL” en la página 12

### Consulta relacionada

“Matrices” en la página 74

“call” en la página 563

“Formulario de entrada” en la página 736

“Registro de entrada” en la página 736

“Convenios de denominación” en la página 672

“pfKeyEquate” en la página 686

“Tipos primitivos” en la página 34

“print” en la página 632

“Componente de programa en formato fuente EGL” en la página 728

“show” en la página 646

“Propiedades de elementos SQL” en la página 67

“connectionService()” en la página 916

“getVAGSysType()” en la página 919

“handleSysLibraryErrors” en la página 947

“segmentedMode” en la página 925

“transfer” en la página 646

### Propiedades y campos de ConsoleField

Las propiedades siguientes son obligatorias en una variable de tipo ConsoleField:

- **fieldLen** (a menos que ConsoleField sea un campo de longitud constante)
- **position**

El campo **name** también es obligatorio, pero no en un ConsoleField de longitud constante.

Las propiedades de ConsoleField son las siguientes:

#### **fieldLen**

Especifica el número de posiciones necesarias para visualizar el valor más grande de interés. Para los consoleFields de longitud constante, no se establece esta propiedad: **fieldLen** es el número de caracteres que ocupa el valor visualizado, tal como se incluye en la propiedad **value**.

**Tipo:** *INT*

**Ejemplo:** *fieldLen = 20*

**Valor predeterminado:** *none*

#### **position**

La ubicación del campo de consola dentro del formulario. La propiedad contiene una matriz de dos enteros positivos: el número de línea seguido del número de columna. El número de línea se calcula desde la parte superior del formulario. De forma parecida, el número de columna se calcula desde la izquierda del formulario.

**Tipo:** *INT[]*

**Ejemplo:** *position = [2, 3]*

**Valor predeterminado:** *[1,1]*

#### **segments**

Especifica la fila, la columna y la longitud de cada *segmento de campo*, que es una subsección de consoleField que puede tener delimitadores. Para crear el aspecto de un recuadro de texto de varias líneas, apile un segmento de campo en líneas sucesivas de la misma columna del formulario, y el conjunto de segmentos actúa como un solo campo.

**Tipo:** *INT[3][]*

**Ejemplo:** *segments = [[5,1,10],[6,1,10]]*

**Valor predeterminado:** *none*

Si se especifica un valor para **segments**, el valor para **position** no se tiene en cuenta, y **fieldLen** debe establecerse en la longitud de todos los segmentos combinados.

Si se especifican múltiples segmentos, el comportamiento de ConsoleField también se ve afectado por el valor del campo **lineWrap**.

#### **validValues**

Especifica la lista de valores que son válidos para la entrada de usuario.

**Tipo:** *Literal de matriz de elementos de uno y dos valores*

**Ejemplo:** *validValues = [ [1,3], 5, 12 ]*

**Valor predeterminado:** *none*

Para obtener detalles, consulte el apartado *validValues*.

Las propiedades de una matriz de `consoleField` incluyen las anteriores (excepto **segments**), así como las siguientes:

#### **columns**

Especifica el número de columnas en las que se deben visualizar los elementos de una matriz de tipo `ConsoleField`. Si, por ejemplo, la matriz tiene cinco elementos y el valor de la propiedad **columns** es dos, la primera línea del formulario muestra dos elementos; la segunda línea muestra dos elementos; y la tercera línea muestra un elemento.

**Tipo:** *INT*

**Ejemplo:** *columns = 3*

**Valor predeterminado:** *1*

Esta propiedad sólo es relevante para matrices de tipo `ConsoleField`. La distribución de los elementos de matriz en la pantalla (tanto horizontal como verticalmente) se ve afectada por la propiedad **orientIndexAcross**.

#### **linesBetweenRows**

Especifica el número de líneas en blanco entre cada línea que contiene un elemento de matriz.

**Tipo:** *INT*

**Ejemplo:** *linesBetweenRows = 3*

**Valor predeterminado:** *0*

Esta propiedad sólo es relevante para matrices de tipo `ConsoleField`.

#### **orientIndexAcross**

Indica si los elementos de matriz se distribuyen horizontalmente en la pantalla, como se muestra en un ejemplo que aparece más adelante.

**Tipo:** *Boolean*

**Ejemplo:** *orientIndexAcross = yes*

**Valor predeterminado:** *yes*

Esta propiedad sólo es relevante para matrices de tipo `consoleField`.

Si la propiedad **orientIndexAcross** está establecida en *yes*, los elementos sucesivos de la matriz se visualizan de izquierda a derecha. En el siguiente ejemplo de dos columnas, cada elemento sucesivo muestra un entero que es equivalente al índice del elemento:

```
1 2
3 4
5
```

Si la propiedad **orientIndexAcross** está establecida en *no*, los elementos sucesivos se visualizan de arriba a abajo:

```
1 4
2 5
3
```

#### **spacesBetweenColumns**

Especifica el número de espacios que separan cada columna de campos.

**Tipo:** *INT*

**Ejemplo:** *spacesBetweenColumns = 3*

**Valor predeterminado:** *1*

Esta propiedad sólo es válida para matrices de tipo `consoleField`.

Los campos de ConsoleField son los siguientes:

### **align**

El campo **align** especifica la posición de datos en un campo de longitud variable cuando la longitud de los datos es menor que la longitud del campo.

**Tipo:** *AlignKind*

**Ejemplo:** *align = left*

**Valor predeterminado:** *left para datos de tipo carácter o de indicación de la hora, right para datos numéricos*

**¿Actualizable durante la ejecución?** *Sí*

Los valores son los siguientes:

### **left**

Colocar los datos a la izquierda del campo. Los espacios iniciales se colocan al final del campo.

### **none**

No justificar los datos. Este valor solamente es válido para los datos de tipo carácter.

### **right**

Colocar los datos a la derecha del campo. Los espacios finales se colocan al principio del campo.

### **autonext**

Indica si, después de que el usuario ha rellenado el ConsoleField actual, el cursor va al campo siguiente.

**Tipo:** *Boolean*

**Ejemplo:** *autonext = yes*

**Valor predeterminado:** *None*

**¿Actualizable durante la ejecución?** *Sí*

El orden de tabulación determina cuál es el siguiente ConsoleField, como se describe en el apartado *Componentes de ConsoleUI y variables relacionadas*.

### **binding**

Especifica el nombre de la variable a la que está enlazado el ConsoleField de forma predeterminada.

**Tipo:** *String*

**Ejemplo:** *binding = "myVar"*

**Valor predeterminado:** *None*

**¿Actualizable durante la ejecución?** *No.*

Para obtener una visión general de los enlaces, consulte el apartado *Componentes de ConsoleUI y variables relacionadas*.

### **caseFormat**

Especifica cómo tratar la entrada y la salida en relación con la distinción entre mayúsculas y minúsculas.

**Tipo:** *CaseFormatKind*

**Ejemplo:** *caseFormat = lowerCase*

**Valor predeterminado:** *defaultCase*

**¿Actualizable durante la ejecución?** *Sí*

Los valores son los siguientes:

**defaultCase (valor predeterminado)**

No tiene ningún efecto sobre las mayúsculas y minúsculas

**lowerCase**

Transforma los caracteres a minúsculas, siempre que sea posible

**upperCase**

Transforma los caracteres a mayúsculas, siempre que sea posible

**color**

Especifica el color del texto en el ConsoleField.

**Tipo:** *ColorKind*

**Ejemplo:** *color = red*

**Valor predeterminado:** *white*

**¿Actualizable durante la ejecución?** *Sí, pero la actualización sólo tiene un efecto visual si el ConsoleField se visualiza (u obtiene el foco) después de que se actualice el campo*

Los valores son los siguientes:

**defaultColor o white (valor predeterminado)**

Blanco

**black**

Negro

**blue**

Azul

**cyan**

Cian

**green**

Verde

**magenta**

Magenta

**red**

Rojo

**yellow**

Amarillo

**comment**

Especifica el *comentario*, que es el texto que se visualiza en la línea de comentario específica de Window (si existe) cuando el cursor está en el ConsoleField.

**Tipo:** *String*

**Ejemplo:** *"Employee name"*

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *No*

**commentKey**

Especifica una tecla que se utiliza para buscar el paquete de recursos que incluye el *comentario*, que es el texto que se visualiza en la línea de comentario específica de Window (si existe) cuando el cursor está en el ConsoleField. Si especifica a la vez **comment** y **commentKey**, se utiliza **comment**.

**Tipo:** *String*

**Ejemplo:** *commentKey = "myKey"*

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *No*

El paquete de recursos se identifica mediante la variable de sistema **ConsoleLib.messageResource**, tal como se describe en *messageResource*.

### **dataType**

Especifica una serie para identificar un tipo de datos. El valor se utiliza para validar que la entrada de usuario (como por ejemplo = 1.5) es compatible con una determinada clase de columna SQL. El campo sólo es relevante cuando la sentencia **openUI** para el ConsoleField (o ConsoleForm relacionado) incluye la propiedad **isConstruct** de la sentencia.

**Tipo:** *String*

**Ejemplo:** *dataType = "NUMBER"*

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *No*

En relación con una entrada numérica, especifique el valor "NUMBER" si permite que el usuario especifique un valor de coma flotante (en cuyo caso, > 1.5 es una entrada de usuario válida); en caso contrario, especifique la serie equivalente de un entero; por ejemplo, "INT".

### **dateFormat**

Indica cómo dar formato a la salida; pero sólo especifique **dateFormat** si el ConsoleField acepta una fecha.

**Tipo:** *String o una constante del sistema relacionada con la fecha*

**Ejemplo:** *dateFormat = isoDateFormat*

**Valor predeterminado:** *none*

**¿Actualizable durante la ejecución?** *No*

Los valores válidos son los siguientes:

*"patrón"*

El valor de *patrón* consiste en un conjunto de caracteres, tal como se describe en la sección *Especificadores de formato de fecha, hora e indicación de la hora*.

Los caracteres pueden eliminarse desde el inicio o el final de una especificación de fecha completa, pero no desde el medio.

### **defaultDateFormat**

El formato de fecha especificado en el entorno local de ejecución de Java.

### **isoDateFormat**

El patrón "aaaa-MM-dd", que es el formato de fecha especificado por International Standards Organization (ISO).

### **usaDateFormat**

El patrón "MM/dd/aaaa", que es el formato de fecha estándar para EE.UU. de IBM.

### **eurDateFormat**

El patrón "dd.MM.aaaa", que es el formato de fecha estándar europeo de IBM.

### **jisDateFormat**

El patrón "aaaa-MM-dd" que es el formato de fecha estándar industrial japonés.

### **systemGregorianCalendar**

Un patrón de 8 ó 10 caracteres que incluye dd (para día del mes numérico), MM (para mes numérico) y aa o aaaa (para año numérico), con caracteres que no sean d, M, a ni dígitos utilizados como separadores.

El formato está en la siguiente propiedad de entorno de ejecución Java:

```
vgj.datemask.gregorian.long.NLS
```

#### *NLS*

El código NLS (soporte de idioma nacional) especificado en la propiedad de entorno de ejecución Java **vgj.nls.code**. El código es uno de los que se listan en targetNLS. Inglés en mayúsculas (código ENP) no está soportado.

Para obtener más detalles acerca de **vgj.nls.code**, consulte el apartado *Propiedades de entorno de ejecución Java (detalles)*.

### **systemJulianDateFormat**

Un patrón de 6 u 8 caracteres que incluye DDD (para día del mes numérico) y aa o aaaa (para año numérico), con caracteres que no sean D, y ni dígitos como separadores.

El formato está en la siguiente propiedad de entorno de ejecución Java:

```
vgj.datemask.julian.long.NLS
```

#### *NLS*

El código NLS (soporte de idioma nacional) especificado en la propiedad de entorno de ejecución Java **vgj.nls.code**. El código es uno de los que se listan en targetNLS. Inglés en mayúsculas (código ENP) no está soportado.

Para obtener más detalles acerca de **vgj.nls.code**, consulte el apartado *Propiedades de entorno de ejecución Java (detalles)*.

### **editor**

Especifica el programa para la interacción de usuario con los los datos; pero sólo es relevante si el ConsoleField está enlazado a una variable de tipo LOB.

**Tipo:** *String*

**Ejemplo:** *editor = "/bin/vi"*

**Valor predeterminado:** *none*

**¿Actualizable durante la ejecución?** *Sí*

Puede especificar el nombre de un ejecutable que se encuentra en la PATH o LIBPATH; como alternativa, puede especificar la vía de acceso totalmente calificada de dicho ejecutable.

### **help**

Especifica el texto que debe visualizarse cuando se produce la siguiente situación:

- El cursor está en el ConsoleField; y
- El usuario pulsa la tecla identificada en **ConsoleLib.key\_help**.

**Tipo:** *String*

**Ejemplo:** *help = "Update the value"*

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *Sí*

### **helpKey**

Especifica una tecla de acceso para buscar el paquete de recursos que contiene el texto que debe visualizarse cuando se produce la siguiente situación:

- El cursor está en el `ConsoleField`; y
- El usuario pulsa la tecla identificada en `ConsoleLib.key_help`.

Si se especifica a la vez `help` y `helpKey`, se utiliza `help`.

**Tipo:** *String*

**Ejemplo:** `helpKey = "myKey"`

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *Sí*

El paquete de recursos se identifica mediante la variable de sistema `ConsoleLib.messageResource`, tal como se describe en *messageResource*.

## **highlight**

Especifica los efectos especiales (si los hay) que se utilizan al visualizar el `ConsoleField`.

**Tipo:** *HighlightKind[]*

**Ejemplo:** `highlight = [reverse, underline]`

**Valor predeterminado:** *[noHighLight]*

**¿Actualizable durante la ejecución?** *Sí, pero la actualización sólo tiene un efecto visual si el `ConsoleField` se visualiza (u obtiene el foco) después de que se actualice el campo `highlight`*

Los valores son los siguientes:

### **noHighlight (valor predeterminado)**

No produce ningún efecto especial. La utilización de este valor altera temporalmente cualquier otro.

### **blink**

No tiene ningún efecto

### **reverse**

Invierte los colores del texto y del fondo, de forma que (por ejemplo), si la pantalla tiene un fondo negro con letras blancas, el fondo pasa a ser negro y el texto pasa a ser blanco.

### **underline**

Coloca un subrayado debajo de las áreas afectadas. El color del subrayado es el color del texto, aunque también se haya especificado el valor `reverse`.

## **initialValue**

Especifica el valor inicial que debe visualizarse.

**Tipo:** *String*

**Ejemplo:** `initialValue = "200"`

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *Sí*

Si la propiedad `setInitial` de la sentencia `openUI` está establecida en `true`, se utiliza el valor de la propiedad `initialValue` del `consoleField`. Sin embargo, si dicha propiedad `openUI` es `false`, se mostrarán los valores actuales de las variables enlazadas y no se tendrá en cuenta el valor de la propiedad `initialValue`.

## **initialValueKey**

Especifica una tecla de acceso para buscar el paquete de recursos que contiene el valor inicial que debe visualizarse. Si se especifica a la vez `initialValue` e `initialValueKey`, se utiliza `initialValue`.

**Tipo:** *String*

**Ejemplo:** *initialValueKey = "myKey"*

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *Sí*

El paquete de recursos se identifica mediante la variable de sistema **ConsoleLib.messageResource**, tal como se describe en *messageResource*.

### **inputRequired**

Indica si se impedirá al usuario alejarse del campo sin introducir un valor.

Si se especifica a la vez **initialValue** e **initialValueKey**, se utiliza **initialValue**.

**Tipo:** *String*

**Ejemplo:** *initialValueKey = "myKey"*

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *No*

El paquete de recursos se identifica mediante la variable de sistema **ConsoleLib.messageResource**, tal como se describe en *messageResource*.

### **intensity**

Especifica la fuerza del font visualizado.

**Tipo:** *IntensityKind[]*

**Ejemplo:** *intensity = [bold]*

**Valor predeterminado:** *[normalIntensity]*

**¿Actualizable durante la ejecución?** *Sí, pero la actualización sólo tiene un efecto visual si el ConsoleField se visualiza (u obtiene el foco) después de que se actualice el campo **intensity***

Los valores son los siguientes:

#### **normalIntensity (valor predeterminado)**

No produce ningún efecto especial. La utilización de este valor altera temporalmente cualquier otro.

#### **bold**

Hace que el texto aparezca en negrita.

#### **dim**

No tiene ningún efecto en este momento. En el futuro, puede hacer que el texto aparezca con menor intensidad, según sea apropiado cuando el campo de entrada esté inhabilitado o se deba quitar el énfasis al mismo.

#### **invisible**

Elimina cualquier indicación de que el campo se encuentra en el formulario.

### **isBoolean**

Indica si el ConsoleField representa un valor booleano. El campo **isBoolean** restringe los valores válidos de ConsoleField y es útil para la entrada o la salida.

El valor de un campo numérico es 0 (falso) o 1 (verdadero).

El valor de un campo de caracteres está representado por una palabra o un subconjunto de una palabra que depende del idioma nacional, y los valores específicos están determinados por el entorno local. En Inglés, por ejemplo, un

campo booleano de tres o más caracteres tiene el valor *yes* (verdadero) o *no* (falso) y el valor de un campo booleano de un carácter tiene el valor truncado *y* o *n*.

**Tipo:** *Boolean*

**Ejemplo:** *isBoolean = yes*

**Valor predeterminado:** *no*

**¿Actualizable durante la ejecución?** *No*

### **lineWrap**

Indica cómo acomodar el texto en una línea nueva siempre que la acomodación es necesaria para no truncar el texto.

**Tipo:** *LineWrapType*

**Ejemplo:** *value = compress*

**Valor predeterminado:** *character*

**¿Actualizable durante la ejecución?** *Sí*

Los valores son los siguientes:

#### **character (el valor por omisión)**

El texto de un campo no se dividirá en un espacio en blanco, sino en la posición de carácter donde está el límite del segmento de campo.

#### **compress**

Si es posible, el texto se dividirá en un espacio en blanco. Cuando el usuario abandone el `consoleField` (navegando a otro `consoleField` o pulsando `Esc`), el valor se asignará a la variable enlazada y se eliminarán los espacios adicionales utilizados para acomodar el texto.

#### **word**

Si es posible, el texto de un campo se dividirá en un espacio en blanco. Cuando el valor se asigna a la variable enlazada, se incluyen espacios adicionales para reflejar cómo se ha rellenado el valor para acomodarse a los límites de palabra.

El campo **lineWrap** sólo es relevante para un `ConsoleField` que tiene múltiples segmentos, ya que está controlado por la propiedad **segments**.

### **masked**

Indica si cada carácter del `ConsoleField` se visualiza en forma de asterisco (\*), según sea apropiado cuando el usuario escribe un contraseña.

**Tipo:** *Boolean*

**Ejemplo:** *masked = yes*

**Valor predeterminado:** *no*

**¿Actualizable durante la ejecución?** *Sí*

### **minimumInput**

Indica el número mínimo de caracteres en la entrada válida.

**Tipo:** *INT*

**Ejemplo:** *minimumInput = 4*

**Valor predeterminado:** *no*

**¿Actualizable durante la ejecución?** *No*

### **name**

Nombre de `ConsoleField`, tal como se utiliza en un contexto de programación

en que el nombre se resuelve durante la ejecución. Se recomienda encarecidamente que el valor del campo `name` sea el mismo que el nombre de la variable.

**Tipo:** *String*

**Ejemplo:** `name = "myField"`

**Valor predeterminado:** *none*

**¿Actualizable durante la ejecución?** *No*

### **numericFormat**

Indica cómo dar formato a la salida; pero sólo especifique **numericFormat** si el `ConsoleField` acepta un número.

**Tipo:** *String*

**Ejemplo:** `numericFormat = "-###@"`

**Valor predeterminado:** *none*

**¿Actualizable durante la ejecución?** *No*

Los caracteres válidos son los siguientes:

- # Un espacio reservado para un dígito.
- \* Utilice un asterisco (\*) como carácter de relleno para un cero inicial.
- & Utilice un cero como carácter de relleno para un cero inicial.
- # Utilice un espacio como carácter de relleno para un cero inicial.
- < Justifique a la izquierda el número.
- , Utilice un separador numérico dependiente del entorno local a menos que la posición contenga un cero inicial.
- . Utilice una coma decimal dependiente del entorno local.
- Utilice un signo menos (-) para los valores menores que 0; utilice un espacio para los valores mayores o iguales que 0.
- + Utilice un signo menos para los valores menores que 0; utilice un signo más (+) para los valores mayores o iguales que 0.
- ( Preceda los valores negativos con un paréntesis izquierdo , según sea apropiado en la contabilidad.
- ) Coloque un paréntesis derecho después de un valor negativo, según sea apropiado en la contabilidad.
- \$ Preceda el valor con un símbolo de moneda dependiente del entorno local.
- @ Coloque el símbolo de moneda dependiente del entorno local después del valor.

### **pattern**

Especifica el patrón para el formato de entrada y salida si el contenido de `ConsoleField` es de tipo carácter.

**Tipo:** *String*

**Ejemplo:** `pattern = "(###) ###-####"`

**Valor predeterminado:** *none*

**¿Actualizable durante la ejecución?** *No*

Los siguientes caracteres de control están disponibles:

- A es un espacio reservado para letras, y el subconjunto de caracteres que se consideran letras depende del entorno local
- # es un espacio reservado para dígitos numéricos
- X es un espacio reservado para un carácter obligatorio de cualquier clase

Los caracteres distintos a los tres anteriores se incluyen en la entrada o salida; pero para la salida, los caracteres recubiertos se pierden:

- Si el patrón de salida es "(###) ###-####", el valor "6219655561212" se muestra como se indica a continuación:

(219) 555-1212

Cada 6 del valor original no está disponible al usuario y se pierde si se actualiza el almacén de datos.

- Para la entrada, el cursor pasa por alto los caracteres de tipo literal y sólo permite escribir donde existen los caracteres de espacio reservado. En el ejemplo actual, si el usuario escribe 2195551212, la serie "(219) 555-1212" se convierte en el valor dentro del ConsoleField y es el valor que se cocina en la variable enlazada.

### **protect**

Especifica si el ConsoleField está protegido contra la actualización de usuario.

**Tipo:** *Boolean*

**Ejemplo:** *protect = yes*

**Valor predeterminado:** *no*

**¿Actualizable durante la ejecución?** *No*

Los valores son los siguientes:

#### **No (el valor por omisión)**

Establece el campo de forma que el usuario pueda sobrescribir el valor en él.

#### **Sí**

Establece el consoleField de forma que el usuario no pueda sobrescribir el valor en él. Además, el cursor pasa por alto el consoleField siempre que el usuario intenta navegar al mismo, con en los casos siguientes:

- El usuario trabaja en el consoleField anterior por orden de tabulación y (a) pulsa el **tabulador** o (b) rellena ese consoleField anterior con contenido cuando el campo **autonext** está establecido en yes.
- El usuario trabaja en el consoleField siguiente por orden de tabulación y pulsa **Mayúsculas Tabulador**.
- El usuario utiliza las teclas de flecha para ir al consoleField siguiente o anterior.

Puede enlazar una variable a un consoleField que está protegido o no. El valor de la propiedad **setInitial** de openUI determina si se visualiza el valor de la variable enlazada.

Se produce un error de tiempo de ejecución si el programa intenta ir a un consoleField que está protegido.

### **SQLColumnName**

Especifica el nombre de la columna de tabla de base de datos que está asociada con el ConsoleField. El nombre se utiliza para crear criterios de búsqueda cuando la sentencia **openUI** para el ConsoleField (o ConsoleForm relacionado) incluye la propiedad **isConstruct** de la sentencia.

**Tipo:** *String*

**Ejemplo:** *SQLColumnName = "ID"*

**Valor predeterminado:** *none*

**¿Actualizable durante la ejecución?** *Sí*

### **timeFormat**

Indica cómo dar formato a la salida; pero sólo especifique **timeFormat** si el ConsoleField acepta una hora.

**Tipo:** *String o una constante del sistema relacionada con la hora*

**Ejemplo:** *timeFormat = isoTimeFormat*

**Valor predeterminado:** *none*

**¿Actualizable durante la ejecución?** *No*

Los valores válidos son los siguientes:

*"patrón"*

El valor de *patrón* consiste en un conjunto de caracteres, tal como se describe en la sección *Especificadores de formato de fecha, hora e indicación de la hora*.

Los caracteres pueden eliminarse desde el inicio o el final de una especificación de hora completa, pero no desde el medio.

### **defaultTimeFormat**

El formato de hora especificado en el entorno local de ejecución de Java.

### **isoTimeFormat**

El patrón "HH.mm.ss", que es el formato de hora especificado por International Standards Organization (ISO).

### **usaTimeFormat**

El patrón "hh:mm AM", que es el formato de hora estándar de EE.UU. de IBM.

### **eurTimeFormat**

El patrón "HH.mm.ss", que es el formato de hora estándar europeo de IBM.

### **jisTimeFormat**

El patrón "HH:mm:ss", que es el formato de hora estándar industrial japonés.

### **timestampFormat**

Indica cómo dar formato a la salida; pero sólo especifique **timestampFormat** si el ConsoleField acepta una indicación de la hora.

**Tipo:** *String o una constante del sistema relacionada con la indicación de la hora*

**Ejemplo:** *timestampFormat = odbcTimestampFormat*

**Valor predeterminado:** *none*

**¿Actualizable durante la ejecución?** *No*

Los valores válidos son los siguientes:

*"patrón"*

El valor de *patrón* consiste en un conjunto de caracteres, tal como se describe en la sección *Especificadores de formato de fecha, hora e indicación de la hora*.

Los caracteres pueden eliminarse desde el inicio o el final de una especificación de indicación de la hora completa, pero no desde el medio.

#### **defaultTimestampFormat**

El formato de indicación de la hora especificado en el entorno local de ejecución de Java.

#### **db2TimeStampFormat**

El patrón "aaaa-MM-dd-HH.mm.ss.ffffff", que es el formato de indicación de la hora predeterminado de IBM DB2.

#### **odbcTimestampFormat**

El patrón "aaaa-MM-dd HH:mm:ss.ffffff", que es el formato de indicación de la hora de ODBC.

#### **value**

El valor actual visualizado en el consoleField. El código puede establecer este valor de modo que la invocación de **ConsoleLib.displayForm** visualiza el valor especificado en el consoleField.

**Tipo:** *String*

**Ejemplo:** *value = "View"*

**Valor predeterminado:** *none*

**¿Actualizable durante la ejecución?** *Sí*

#### **verify**

Indica si se solicita al usuario que vuelva a escribir el mismo valor después de intentar salir del ConsoleField.

**Tipo:** *String*

**Ejemplo:** *value = "View"*

**Valor predeterminado:** *none*

**¿Actualizable durante la ejecución?** *Sí*

Los valores son los siguientes:

#### **No (el valor por omisión)**

El entorno de ejecución EGL no emite una solicitud especial.

#### **Sí**

Cuando el usuario intenta abandonar el ConsoleField, el entorno de ejecución EGL actúa de la manera siguiente:

- Borra el consoleField, manteniendo allí el cursor
- Muestra un mensaje para que el usuario repita la entrada
- Compara los dos valores de entrada cuando el usuario intenta volver a abandonar el consoleField

Si los valores coinciden, la variable enlazada recibe dicho valor y el proceso continúa de la forma habitual. Si los valores no coinciden, el contenido de consoleField vuelve al valor que precedía a la primera de las dos entradas de usuario, y el cursor permanece en el campo.

#### **Conceptos relacionados**

"Interfaz de usuario de consola" en la página 177

#### **Consulta relacionada**

"Componentes de ConsoleUI y variables relacionadas" en la página 180

"Especificadores de fecha, hora e indicación de la hora" en la página 45

"Propiedades de ejecución de Java (detalles)" en la página 540 "openUI" en la página 620

"validValues" en la página 720

### Tarea relacionada

“Crear una interfaz con consoleUI” en la página 178

## Propiedades de ConsoleForm en consoleUI de EGL

Las propiedades de un componente de registro de tipo ConsoleForm son las siguientes, y sólo formSize es obligatorio:

### delimiters

Especifica los caracteres que se visualizan antes y después de los campos de entrada. Los caracteres sólo se visualizan si el valor de la propiedad **showBrackets** es *yes*.

**Tipo:** *Literal de tipo String*

**Ejemplo:** `delimiters = "<>/"`

**Valor predeterminado:** `"[]|"`

Siempre que sea posible, el primer carácter se visualiza antes de cada ConsoleField de longitud no constante, y el segundo carácter se visualiza después de cada ConsoleField de longitud no constante. Sin embargo, el tercer carácter se visualiza entre dos ConsoleFields de longitud no constante separados por una única posición.

Si se especifican menos de tres caracteres, se aplica un carácter predeterminado para cada carácter no especificado. Si se especifican más de tres caracteres, el cuarto carácter y los siguientes no se tienen en cuenta.

### formSize

Las dimensiones del formulario. El campo debe contener una matriz de dos enteros positivos: el número de líneas seguido del número de columnas.

**Tipo:** *INT[2]*

**Ejemplo:** `size = [24, 80]`

**Valor predeterminado:** *none*

Si una de las dimensiones sobrepasa el tamaño de la ventana donde se visualiza el formulario, se reduce el tamaño del formulario para que quepa en las dimensiones de la ventana. Sin embargo, si un ConsoleField no cabe en el ventana, el programa finaliza.

### name

Nombre de formulario, tal como se utiliza en un contexto de programación en que el nombre se resuelve durante la ejecución. Se recomienda que el valor del campo name, si existe, sea el mismo que el nombre de la variable.

**Tipo:** *String*

**Ejemplo:** `name = "myForm"`

**Valor predeterminado:** *none*

El campo name se utiliza en funciones del sistema como, por ejemplo, **ConsoleLib.displayFormByName**.

### showBrackets

Indica si los ConsoleFields de longitud no constante están delimitados por un par de caracteres, como por ejemplo corchetes.

**Tipo:** *Boolean*

**Ejemplo:** `showBrackets = no`

**Valor predeterminado:** *yes*

Para obtener otros detalles, consulte la propiedad **delimiters**.

### Conceptos relacionados

“Interfaz de usuario de consola” en la página 177

### Consulta relacionada

“Componentes de ConsoleUI y variables relacionadas” en la página 180

“openUI” en la página 620

### Tarea relacionada

“Crear una interfaz con consoleUI” en la página 178

## Campos de Menu en consoleUI de EGL

La lista siguiente define los campos en una variable de tipo Menu. Debe especificar el campo **labelText** o **labelTextKey**.

### labelText

La etiqueta que se visualiza a la izquierda de la lista de menuItems.

**Tipo:** *Literal de tipo String*

**Ejemplo:** *labelText = "Options: "*.

**Valor predeterminado:** *none*.

**¿Actualizable durante la ejecución?** *No*

### labelKey

Especifica una tecla para buscar el paquete de recursos que contiene la etiqueta de menú. Si especifica a la vez **labelText** y **labelKey**, se utiliza **labelText**.

**Tipo:** *String*

**Ejemplo:** *labelKey = "myKey"*

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *No*

El paquete de recursos se identifica mediante la variable de sistema **ConsoleLib.messageResource**, tal como se describe en *messageResource*.

### menuItems

Una matriz de elementos de menú, cada uno de los cuales se declara en el programa o se crea dinámicamente con la palabra clave **new**. Para obtener detalles sobre la segunda opción, consulte el apartado *Utilización de new en consoleUI*.

**Tipo:** *MenuItem[]*

**Ejemplo:** *menuItems = [myItem, new MenuItem {name = "Remove", labelText = "Delete all"}]*.

**Valor predeterminado:** *none*.

**¿Actualizable durante la ejecución?** *No*

Puede añadir un menuItem en el programa utilizando la sintaxis siguiente:

```
miMenu.MenuItems.addElement(miMenuItem)
```

*miMenu*

Nombre de la variable de tipo Menu.

*miMenuItem*

Nombre de la variable de tipo MenuItem.

El programa finaliza si se emite una sentencia **openUI** para un menú en el que no existen menuItems.

### Conceptos relacionados

“Interfaz de usuario de consola” en la página 177

### Consulta relacionada

“Matrices” en la página 74

“Componentes de ConsoleUI y variables relacionadas” en la página 180

“openUI” en la página 620

“Campos de MenuItem en consoleUI de EGL”

“Utilización de new en ConsoleUI” en la página 183

### Tarea relacionada

“Crear una interfaz con consoleUI” en la página 178

## Campos de MenuItem en consoleUI de EGL

La lista siguiente define los consoleFields en una variable de tipo MenuItem. Ninguno de los consoleFields es obligatorio; puede determinar la selección del usuario estableciendo cualquiera de estos tres campos: **accelerators**, **labelText** o **labelKey**.

### accelerators

Indica las pulsaciones de tecla que son equivalentes a la selección del usuario del menuItem. Cada una de estas pulsaciones de tecla hace que se ejecute la cláusula OnEvent de la sentencia **openUI** correspondiente a la selección de menuItem.

**Tipo:** *String[]*

**Ejemplo:** *accelerators = ["F1", "ALT\_F1"]*

**Valor predeterminado:** *none*

**¿Actualizable durante la ejecución?** *No*

### comment

Especifica el *comentario*, que es el texto que se visualiza en la línea de comentario específica de menuItem cuando se selecciona el menuItem.

**Tipo:** *String*

**Ejemplo:** *"Delete the record"*

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *Sí*

La línea de comentario es la que aparece debajo de la línea de menú.

### commentKey

Especifica un tecla utilizada para buscar el paquete de recursos que incluye el *comentario*, que es el texto que se visualiza en la línea de comentario específica de menuItem (si existe) cuando se selecciona el menuItem. Si especifica a la vez **comment** y **commentKey**, se utiliza **comment**.

**Tipo:** *String*

**Ejemplo:** *commentKey = "myKey"*

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *Sí*

El paquete de recursos se identifica mediante la variable de sistema **ConsoleLib.messageResource**, tal como se describe en *messageResource*.

### help

Especifica el texto que debe visualizarse cuando se produce la siguiente situación:

- El menuItem está seleccionado; y
- El usuario pulsa la tecla identificada en **ConsoleLib.key\_help**.

**Tipo:** *String*

**Ejemplo:** *help = "Deletion is permanent"*

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *Sí*

### **helpKey**

Especifica una tecla de acceso para buscar el paquete de recursos que contiene el texto que debe visualizarse cuando se produce la siguiente situación:

- El menuItem está seleccionado; y
- El usuario pulsa la tecla identificada en **ConsoleLib.key\_help**.

Si se especifica a la vez **help** y **helpKey**, se utiliza **help**.

**Tipo:** *String*

**Ejemplo:** *helpKey = "myKey"*

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *Sí*

El paquete de recursos se identifica mediante la variable de sistema **ConsoleLib.messageResource**, tal como se describe en *messageResource*.

### **labelText**

La etiqueta que representa el menuItem.

**Tipo:** *Literal de tipo String*

**Ejemplo:** *labelText = "Delete"*.

**Valor predeterminado:** *none*.

**¿Actualizable durante la ejecución?** *No*

### **labelKey**

Especifica una tecla para buscar el paquete de recursos que contiene la etiqueta de menuItem. Si especifica a la vez **labelText** y **labelKey**, se utiliza **labelText**.

**Tipo:** *String*

**Ejemplo:** *labelKey = "myKey"*

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *No*

El paquete de recursos se identifica mediante la variable de sistema **ConsoleLib.messageResource**, tal como se describe en *messageResource*.

### **name**

Nombre de menuItem, tal como se utiliza en un contexto de programación en que el nombre se resuelve durante la ejecución. Concretamente, el nombre se utiliza en la sentencia **openUI** que responde a la selección de menuItem.

Se recomienda que el valor del campo name sea el mismo que el nombre de la variable.

**Tipo:** *String*

**Ejemplo:** *name = "myItem"*

**Valor predeterminado:** *none*

**¿Actualizable durante la ejecución?** *No*

### Conceptos relacionados

“Interfaz de usuario de consola” en la página 177

### Consulta relacionada

“Componentes de ConsoleUI y variables relacionadas” en la página 180

“Campos de Menu en consoleUI de EGL” en la página 455

“openUI” en la página 620

### Tarea relacionada

“Crear una interfaz con consoleUI” en la página 178

## Campos de PresentationAttributes en consoleUI de EGL

La lista siguiente define los campos que pueden establecerse o recuperarse en cualquier variable de sistema de tipo PresentationAttributes:

### color

Especifica un color.

**Tipo:** *ColorKind*

**Ejemplo:** *color = red*

**Valor predeterminado:** *white*

**¿Actualizable durante la ejecución?** *Sí, pero la actualización sólo tiene un efecto visual para la salida que se visualiza después de que se actualice el campo color*

Los valores son los siguientes:

#### **defaultColor o white (valor predeterminado)**

Blanco

#### **black**

Negro

#### **blue**

Azul

#### **cyan**

Cian

#### **green**

Verde

#### **magenta**

Magenta

#### **red**

Rojo

#### **yellow**

Amarillo

### highlight

Especifica los efectos especiales (si los hay) que se utilizan al visualizar la salida.

**Tipo:** *HighlightKind[]*

**Ejemplo:** *highlight = [reverse, underline]*

**Valor predeterminado:** *[noHighLight]*

*¿Actualizable durante la ejecución? Sí, pero la actualización sólo tiene un efecto visual para la salida que se visualiza después de que se actualice el campo highlight*

Los valores son los siguientes:

**noHighlight (valor predeterminado)**

No produce ningún efecto especial. La utilización de este valor altera temporalmente cualquier otro.

**blink**

No tiene ningún efecto en este momento.

**reverse**

Invierte los colores del texto y del fondo, de forma que (por ejemplo), si la pantalla tiene un fondo negro con letras blancas, el fondo pasa a ser negro y el texto pasa a ser blanco.

**underline**

Coloca un subrayado debajo de las áreas afectadas. El color del subrayado es el color del texto, aunque también se haya especificado el valor **reverse**.

**intensity**

Especifica la fuerza del font visualizado.

**Tipo:** *IntensityKind[]*

**Ejemplo:** *intensity = [bold]*

**Valor predeterminado:** *[normalIntensity]*

*¿Actualizable durante la ejecución? Sí, pero la actualización sólo tiene un efecto visual para la salida que se visualiza después de que se actualice el campo intensity*

Los valores son los siguientes:

**normalIntensity (valor predeterminado)**

No produce ningún efecto especial. La utilización de este valor altera temporalmente cualquier otro.

**bold**

Hace que el texto aparezca en negrita.

**dim**

No tiene ningún efecto en este momento. En el futuro, puede hacer que el texto aparezca con menor intensidad, según sea apropiado cuando todos los campos de entrada estén inhabilitados.

**invisible**

Elimina cualquier indicación de que el texto se encuentra en el formulario.

**Conceptos relacionados**

“Interfaz de usuario de consola” en la página 177

**Consulta relacionada**

“currentDisplayAttrs” en la página 769

“currentRowAttrs” en la página 769

“defaultDisplayAttributes” en la página 769

“defaultInputAttributes” en la página 770

“Componentes de ConsoleUI y variables relacionadas” en la página 180

“openUI” en la página 620

## Tarea relacionada

“Crear una interfaz con consoleUI” en la página 178

## Campos de Prompt en consoleUI de EGL

La lista siguiente define los campos en una variable de tipo Prompt. Ninguno de los campos es obligatorio.

### isChar

Indica si, después que se visualice la solicitud, la primera pulsación de tecla del usuario finaliza la operación.

**Tipo:** *Boolean*

**Ejemplo:** *isChar = yes*

**Valor predeterminado:** *no*

**¿Actualizable durante la ejecución?** *Sí*

Los valores son los siguientes:

#### **no (valor predeterminado)**

La operación finaliza cuando el usuario pulsa **Intro** o cuando pulsa una tecla asociada a una cláusula OnEvent de la sentencia **openUI** que muestra la solicitud. La variable a la que está enlazada la solicitud recibe los caracteres de entrada.

#### **yes**

La primera pulsación de tecla del usuario finaliza la operación. La variable a la que está enlazada la solicitud recibe el carácter, si el carácter es imprimible.

En cualquier caso, puede responder a una determinada pulsación de tecla estableciendo una cláusula OnEvent de tipo ON\_KEY.

### message

Especifica el texto de la solicitud para el usuario.

**Tipo:** *String*

**Ejemplo:** *message = "Type here: "*

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *Sí, antes de que el código emita la sentencia **openUI***

### messageKey

Especifica una tecla que se utiliza para buscar el paquete de recursos que incluye el texto de la solicitud. Si se especifica a la vez **message** y **messageKey**, se utiliza **message**.

**Tipo:** *String*

**Ejemplo:** *messageKey = "promptText"*

**Valor predeterminado:** *Serie vacía*

**¿Actualizable durante la ejecución?** *Sí*

El paquete de recursos se identifica mediante la variable de sistema **ConsoleLib.messageResource**, tal como se describe en *messageResource*.

### responseAttr

Especifica los atributos de presentación que se utilizan al visualizar la entrada de usuario.

**Tipo:** *Literal de PresentationAttributes*

**Ejemplo:** *responseAttr {color = green, highlight = [underline], intensity = [bold]}*

**Valor predeterminado:** *no*

**¿Actualizable durante la ejecución?** *Sí*

Este campo sólo tiene un efecto si el campo **isChar** está establecido en *no*.

Para obtener información detallada sobre los valores de **responseAttr**, consulte el apartado *Campos de PresentationAttributes en consoleUI de EGL*.

#### **Conceptos relacionados**

“Interfaz de usuario de consola” en la página 177

#### **Consulta relacionada**

“Componentes de ConsoleUI y variables relacionadas” en la página 180

“Propiedades de ejecución de Java (detalles)” en la página 540

“messageResource” en la página 783

“openUI” en la página 620

“Campos de PresentationAttributes en consoleUI de EGL” en la página 458

#### **Tarea relacionada**

“Crear una interfaz con consoleUI” en la página 178

## **Campos de Window en consoleUI de EGL**

La lista siguiente define los campos en una variable de tipo Window. Ninguno de los campos es obligatorio, pero **size** se necesita a efectos prácticos.

#### **color**

Especifica el color que se utiliza al visualizar las siguientes clases de salida en la ventana:

- Etiquetas en consoleForms
- Campos de entrada en solicitudes
- Borde de ventana
- Salida de funciones de sistema, como por ejemplo

#### **ConsoleLib.displayAtPosition**

**Tipo:** *ColorKind*

**Ejemplo:** *color = red*

**Valor predeterminado:** *white*

**¿Actualizable durante la ejecución?** *Sí, pero la actualización sólo tiene un efecto visual si la ventana se abre después de que se actualice el campo*

Los valores son los siguientes:

#### **defaultColor o white (valor predeterminado)**

Blanco

#### **black**

Negro

#### **blue**

Azul

#### **cyan**

Cian

#### **green**

Verde

#### **magenta**

Magenta

**red**

Rojo

**yellow**

Amarillo

### **commentLine**

Establece el número de la línea en la que se visualiza un comentario (si existe) si el campo **hasCommentLine** de Window está establecido en *yes*. El número de línea se calcula desde la parte superior del área de contenido de la ventana de consola (en cuyo caso la primera línea es 1), o bien (si el valor es negativo) desde la parte inferior de este área (en cuyo caso la última línea es -1, la penúltima es -2 y así sucesivamente).

**Tipo:** *INT*

**Ejemplo:** *commentLine = 10*

**Valor predeterminado:** *última línea de la ventana (pero si sólo la ventana de pantalla está abierta, el comentario está en la penúltima línea de dicha ventana)*

**¿Actualizable durante la ejecución?** *Sí, pero la actualización sólo tiene un efecto visual si la ventana se abre después de que se actualice el campo*

La validez del valor sólo se determina en tiempo de ejecución.

### **formLine**

Establece el número de la línea en la que se visualizan los formularios. El número de línea se calcula desde la parte superior del área de contenido de la ventana de consola (en cuyo caso la primera línea es 1), o bien (si el valor es negativo) desde la parte inferior de este área (en cuyo caso la última línea es -1, la penúltima es -2 y así sucesivamente).

**Tipo:** *INT*

**Ejemplo:** *formLine = 8*

**Valor predeterminado:** *3*

**¿Actualizable durante la ejecución?** *Sí, pero la actualización sólo tiene un efecto visual si la ventana se visualiza después de que se actualice el campo*

La validez del valor sólo se determina en tiempo de ejecución.

### **hasBorder**

Indica si la ventana está rodeada por un borde. Si el valor es *yes*, el color del borde se especifica en el campo *color* de Window.

**Tipo:** *Boolean*

**Ejemplo:** *hasBorder = yes*

**Valor predeterminado:** *no*

**¿Actualizable durante la ejecución?** *Sí, pero la actualización sólo tiene un efecto visual si la ventana se abre después de que se actualice el campo*

### **hasCommentLine**

Indica si la ventana tiene reservada una línea para *comentarios*, que son entradas de texto que se visualizan cuando el cursor entra en un *consoleField*. Si el valor es *yes*, el número de línea se especifica en el campo *commentLine* de Window.

**Tipo:** *Boolean*

**Ejemplo:** *hasCommentLine = yes*

**Valor predeterminado:** *no*

**¿Actualizable durante la ejecución?** *Sí, pero la actualización sólo tiene un efecto visual si la ventana se abre después de que se actualice el campo*

## highlight

Especifica los efectos especiales (si los hay) que se utilizan al visualizar las siguientes clases de salida en la ventana:

- Etiquetas en consoleForms
- Campos de entrada en solicitudes
- Borde de ventana
- Salida de funciones de sistema, como por ejemplo

### **ConsoleLib.displayAtPosition**

**Tipo:** *HighlightKind[]*

**Ejemplo:** *highlight = [reverse, underline]*

**Valor predeterminado:** *[noHighLight]*

**¿Actualizable durante la ejecución?** *Sí, pero la actualización sólo tiene un efecto visual si la ventana se visualiza después de que se actualice el campo*

Los valores son los siguientes:

### **noHighlight (valor predeterminado)**

No produce ningún efecto especial. La utilización de este valor altera temporalmente cualquier otro.

### **blink**

No tiene ningún efecto en este momento.

### **reverse**

Invierte los colores del texto y del fondo, de forma que (por ejemplo), si la pantalla tiene un fondo negro con letras blancas, el fondo pasa a ser negro y el texto pasa a ser blanco.

### **underline**

Coloca un subrayado debajo de las áreas afectadas. El color del subrayado es el color del texto, aunque se haya invertido el color del texto porque también se haya especificado el valor **Reverse**.

## intensity

Especifica la fuerza del font visualizado que se utiliza al visualizar las siguientes clases de salida en la ventana:

- Etiquetas en consoleForms
- Campos de entrada en solicitudes
- Borde de ventana
- Salida de funciones de sistema, como por ejemplo

### **ConsoleLib.displayAtPosition**

**Tipo:** *IntensityKind[]*

**Ejemplo:** *intensity = [bold]*

**Valor predeterminado:** *[normalIntensity]*

**¿Actualizable durante la ejecución?** *Sí, pero la actualización sólo tiene un efecto visual si la ventana se abre después de que se actualice el campo*

Los valores son los siguientes:

### **normalIntensity (valor predeterminado)**

No produce ningún efecto especial. La utilización de este valor altera temporalmente cualquier otro.

### **bold**

Hace que el texto aparezca en negrita.

**dim**

No tiene ningún efecto en este momento. En el futuro, puede hacer que el texto aparezca con menor intensidad, según sea apropiado cuando todos los campos de entrada estén inhabilitados.

**invisible**

Elimina cualquier indicación de que el campo se encuentra en el formulario.

**menuLine**

Establece el número de la línea en la que se visualiza un menú (si existe) en la ventana. El número de línea se calcula desde la parte superior del área de contenido de la ventana de consola (en cuyo caso la primera línea es 1), o bien (si el valor es negativo) desde la parte inferior de este área (en cuyo caso la última línea es -1, la penúltima es -2 y así sucesivamente).

**Tipo:** *INT*

**Ejemplo:** *menuLine = 2*

**Valor predeterminado:** *1*

**¿Actualizable durante la ejecución?** *Sí, pero la actualización sólo tiene un efecto visual si la ventana se abre después de que se actualice el campo*

La validez del valor sólo se determina en tiempo de ejecución.

**messageLine**

Establece el número de la línea en la que se visualiza un mensaje (si existe) en la ventana. El número de línea se calcula desde la parte superior del área de contenido de la ventana de consola (en cuyo caso la primera línea es 1), o bien (si el valor es negativo) desde la parte inferior de este área (en cuyo caso la última línea es -1, la penúltima es -2 y así sucesivamente).

**Tipo:** *INT*

**Ejemplo:** *messageLine = 3*

**Valor predeterminado:** *2*

**¿Actualizable durante la ejecución?** *Sí, pero la actualización sólo tiene un efecto visual si la ventana se abre después de que se actualice el campo*

La validez del valor sólo se determina en tiempo de ejecución.

**name**

Nombre de ventana, tal como se utiliza en un contexto de programación en que el nombre se resuelve durante la ejecución. Se recomienda que el valor del campo name sea el mismo que el nombre de la variable.

**Tipo:** *String*

**Ejemplo:** *name = "myWindow"*

**Valor predeterminado:** *none*

**¿Actualizable durante la ejecución?** *No*

**position**

La ubicación de la esquina superior izquierda de la ventana dentro del área de contenido de la ventana de pantalla. El campo contiene una matriz de dos enteros: el número de línea seguido del número de columna. El número de línea se calcula desde la parte superior del área de contenido de la ventana de consola (en cuyo caso la primera línea es 1), o bien (si el valor es negativo) desde la parte inferior de este área (en cuyo caso la última línea es -1, la penúltima es -2 y así sucesivamente). El número de columna se calcula desde la izquierda del área de contenido de la ventana de consola, y la primera columna es 1.

**Tipo:** *INT[2]*

**Ejemplo:** *position = [2, 3]*

**Valor predeterminado:** *[1,1]*

**¿Actualizable durante la ejecución?** *No*

### **promptLine**

Establece el número de la línea en la que se visualiza una solicitud (si existe) en la ventana. El número de línea se calcula desde la parte superior del área de contenido de la ventana de consola, o bien (si el valor es negativo) desde la parte inferior de este área.

**Tipo:** *INT*

**Ejemplo:** *promptLine = 4*

**Valor predeterminado:** *1*

**¿Actualizable durante la ejecución?** *Sí, pero la actualización sólo tiene un efecto visual si la ventana se abre después de que se actualice el campo*

La validez del valor sólo se determina en tiempo de ejecución.

### **size**

Una matriz de dos enteros positivos que representan las dimensiones de la ventana: el número de líneas seguido del número de columnas.

**Tipo:** *INT[2]*

**Ejemplo:** *size = [24, 80]*

**Valor predeterminado:** *none*

**¿Actualizable durante la ejecución?** *No*

Se requiere un valor a efectos prácticos. Si se visualiza una ventana que no tiene un valor para **size**, el tiempo de ejecución presenta una ventana que es demasiado pequeña para el contenido.

Si una de las dimensiones sobrepasa el tamaño disponible en el área de contenido de la ventana de pantalla, se produce un error durante la ejecución.

### **Conceptos relacionados**

“Interfaz de usuario de consola” en la página 177

### **Consulta relacionada**

“Componentes de ConsoleUI y variables relacionadas” en la página 180

“openUI” en la página 620

### **Tarea relacionada**

“Crear una interfaz con consoleUI” en la página 178

---

## **containerContextDependent**

La propiedad del componente de función **containerContextDependent** permite ampliar el espacio de nombres que se utiliza para resolver las referencias de función desde el componente de función que incluye la propiedad. Los valores válidos son *no* (el valor por omisión) y *sí*.

Es recomendable evitar la utilización de esta posibilidad al desarrollar código nuevo. La propiedad está disponible principalmente para migrar programas de VisualAge Generator. Sin embargo, si establece esta propiedad en *yes*, las implicaciones son las siguientes:

- Si los pasos usuales de una búsqueda de nombres no resuelve una referencia en tiempo de edición, el editor EGL no marca como errores las referencias no resueltas.
- Si los pasos habituales de una búsqueda de nombres no resuelven una referencia en tiempo de generación, la búsqueda continúa revisando el espacio de nombres del programa, biblioteca o PageHandler que contiene el componente de función.
- Si ha declarado una función en el nivel superior de un archivo fuente EGL en lugar de físicamente dentro de un contenedor (un programa, un PageHandler o una biblioteca), esa función puede invocar funciones de biblioteca sólo si se da la situación siguiente:
  - El contenedor incluye una sentencia use que hace referencia a la biblioteca
  - En la función invocante, la propiedad **containerContextDependent** se establece en *yes*

#### Conceptos relacionados

“Referencias a componentes” en la página 21

#### Consulta relacionada

“Componente de función en formato fuente EGL” en la página 527

]“Declaración use” en la página 954

---

## Autorización de base de datos y nombres de tabla

Un ID de autorización es una serie de caracteres que se pasa al gestor de bases de datos cuando se establece una conexión entre éste y un programa, cuando el programa prepara otro programa o permite al usuario final acceder a tablas SQL. La serie de caracteres es el identificador de usuario necesario para comprobar la autorización de acceso a base de datos que posee el preparador o el usuario final.

El origen del ID de autorización depende del sistema en el que se produzca el acceso a la base de datos.

La situación por lo que respecta a programas Java generados por EGL es la siguiente:

- El ID de autorización lo obtiene el gestor de bases de datos cuando se establece una conexión entre éste y el programa:
  - En relación a la base de datos por omisión, el ID de autorización es el valor especificado para la propiedad de entorno de ejecución Java **vgj.jdbc.default.userid**
  - Al invocar las funciones de sistema **sysLib.connect** o **VGLib.connectionService**, el ID de autorización es el valor especificado para el parámetro `userID`

El ID de autorización puede utilizarse al especificar un nombre de tabla. En ese caso, puede especificar un calificador de nombre de tabla, de acuerdo con la siguiente sintaxis:

*propietarioTabla.miTabla*

*propietarioTabla*

Un calificador conocido por el gestor de bases de datos y que es necesario para identificar la tabla. Durante la creación de la tabla, el calificador es el ID de autorización de la persona que ha creado la tabla.

*miTabla*

El nombre de la tabla.

Para obtener más información acerca de los ID de autorización, consulte la documentación del gestor de bases de datos.

### Conceptos relacionados

“SQL dinámico” en la página 240

“Propiedades de tiempo de ejecución Java” en la página 347

“Soporte de SQL” en la página 229

### Consulta relacionada

“Propiedades de ejecución de Java (detalles)” en la página 540

“Componente de registro SQL en formato fuente EGL” en la página 748

“connect()” en la página 895

“connectionService()” en la página 916

---

## Conversión de datos

Debido a las diferencias en la interpretación de los datos en los diversos entornos de ejecución, es posible que el programa necesite convertir los datos que pasa de un entorno a otro. La conversión de datos se produce durante Java.

El código también utiliza una tabla de conversión en las siguientes situaciones de ejecución:

- El código Java generado por EGL llama a un programa en CICS para z/OS.  
En este caso, puede especificar la tabla de conversión en un elemento callLink que haga referencia al programa llamado. Como alternativa, puede indicar (en ese elemento callLink) que la variable de sistema sysVar.callConversionTable identifique la tabla de conversión durante la ejecución.
- Un programa generado por EGL (en una plataforma que da soporte al juego de caracteres EBCDIC) efectúa una transferencia asíncrona a un programa de una plataforma que da soporte al juego de caracteres ASCII, como puede ocurrir cuando el programa transfiriente invoca la función de sistema sysLib.startTransaction.  
En este caso, puede especificar la tabla de conversión en un elemento asynchLink que haga referencia al programa al que se transfiere el control. Como alternativa, puede indicar (en ese elemento asynchLink) que la variable de sistema sysVar.callConversionTable identifique la tabla de conversión durante la ejecución.
- Un programa Java generado por EGL muestra un formulario de texto o de impresión que incluye series de caracteres árabes o hebreos; o presenta un formulario de texto que acepta una serie de tales caracteres del usuario.  
En estos casos, especifique la tabla de conversión bidireccional en la variable de sistema sysVar.formConversionTable.

Utilizará la conversión en tiempo de ejecución, por ejemplo, si el código coloca valores en uno o dos registro redefinidos, cada uno de los cuales hace referencia a la misma área de memoria que un registro que se pasa a otro programa. Supongamos que las características de los datos que se pasan son diferentes, dependiendo del registro redefinido al que se asignan valores. En este caso, los requisitos de la conversión de datos no pueden conocerse durante la generación.

Las secciones que siguen proporcionan los siguientes detalles:

-

- “Conversión de datos cuando el invocante es código Java”
- “Algoritmo de conversión” en la página 469

## Conversión de datos cuando el invocante es código Java

Las siguientes normas atañen al código Java:

- Cuando un programa o envoltura Java generado invoca un programa Java generado, la conversión se produce en el llamador, de acuerdo con un conjunto de clases EGL invocadas durante la ejecución. En la mayoría de los casos es suficiente con solicitar que no se realice ninguna conversión, aunque el llamador acceda a una plataforma remota que utilice una página de códigos que sea diferente de la página de códigos utilizada por el invocante. Sin embargo, debe especificar una tabla de conversión en los siguientes casos:
  - El llamador es código Java y se encuentra en una máquina que da soporte a una página de códigos
  - El programa llamado no es Java y se encuentra en una máquina que da soporte al otra página de códigos

En este caso, el nombre de tabla es un símbolo que indica el tipo de conversión necesaria durante la ejecución.

- Cuando un programa Java generado accede a una cola de mensajes MQSeries remota, la conversión se produce en el invocante, de acuerdo con un conjunto de clases EGL invocadas durante la ejecución. Si el llamador accede a una plataforma remota que utiliza una página de códigos que es diferente de la página de códigos utilizada por el invocante, especifique una tabla de conversión en el elemento de asociación que hace referencia a la cola de mensajes MQSeries.

La tabla siguiente lista las tablas de conversión a las que puede acceder el código Java generado durante la ejecución. Cada nombre tiene el formato CSOJx:

- x Representa el número de página de códigos de la plataforma invocada. Cada uno de los números se especifica en la publicación *Character Data Representation Architecture Reference and Registry*, SC09-2190. El registro identifica los juegos de caracteres soportados por las tablas de conversión.

Idioma	Plataforma de programa invocado		
	UNIX	Windows 2000/NT/XP	z/OS UNIX System Services o iSeries Java
Árabe	CSOJ1046	CSOJ1256	CSOJ420
Chino simplificado	CSOJ1381	CSOJ1386	CSOJ1388
Chino tradicional	CSOJ950	CSOJ950	CSOJ1371
Cirílico	CSOJ866	CSOJ1251	CSOJ1025
Danés	CSOJ850	CSOJ850	CSOJ277
Europa del Este	CSOJ852	CSOJ1250	CSOJ870
Inglés (Reino Unido)	CSOJ850	CSOJ1252	CSOJ285
Inglés (Estados Unidos)	CSOJ850	CSOJ1252	CSOJ037
Francés	CSOJ850	CSOJ1252	CSOJ297
Alemán	CSOJ850	CSOJ1252	CSOJ273
Griego	CSOJ813	CSOJ1253	CSOJ875

Idioma	Plataforma de programa invocado		
	UNIX	Windows 2000/NT/XP	z/OS UNIX System Services o iSeries Java
Hebreo	CSOJ856	CSOJ1255	CSOJ424
Japonés	CSOJ943	CSOJ943	CSOJ1390 (Katakana SBCS), CSOJ1399 (Latino SBCS)
Coreano	CSOJ949	CSOJ949	CSOJ1364
Portugués	CSOJ850	CSOJ1252	CSOJ037
Español	CSOJ850	CSOJ1252	CSOJ284
Sueco	CSOJ850	CSOJ1252	CSOJ278
Alemán de Suiza	CSOJ850	CSOJ1252	CSOJ500
Turco	CSOJ920	CSOJ1254	CSOJ1026

Si no especifica un valor para la tabla de conversión en el componente de opciones de enlace al llamar a un programa desde Java, las tablas de conversión por omisión serán las correspondientes al Inglés de Estados Unidos.

## Algoritmo de conversión

La conversión de datos de registros y estructuras se basa en las declaraciones de los elementos de estructura que no tienen subestructuras.

Los datos de tipo CHAR, DBCHAR o MBCHAR se convierten de acuerdo con las tablas de conversión Java (para la conversión que se produce en un invocante generado por EGL).

No se realiza ninguna conversión en los elementos de datos de relleno (elementos de datos sin nombre) ni en los elementos de datos de tipo DECIMAL, PACF, HEX o UNICODE.

En la conversión de EBCDIC a ASCII de datos de tipo MBCHAR, la rutina de conversión suprime los caracteres de desplazamiento a teclado ideográfico o a teclado estándar (SO/SI) e inserta un número de blancos equivalente al final del elemento de datos. En la conversión de ASCII a EBCDIC, la rutina de conversión inserta caracteres SO/SI alrededor de las series de doble byte y trunca el valor en el último carácter válido que cabe en el campo. Si el campo MBCHAR es un registro de longitud variable y el final del registro actual se encuentra en el campo MBCHAR, la longitud del registro se ajusta para reflejar la inserción o supresión de los caracteres SO/SI. La longitud del registro indica dónde finaliza el registro actual.

Para los elementos de datos de tipo BIN, la rutina de conversión invierte el orden de los bytes del elemento si la plataforma llamada o llamadora utiliza formato binario Intel y la otra plataforma no lo hace.

En los elementos de datos de tipo NUM o NUMC, la rutina de conversión convierte todos los bytes excepto el último mediante el algoritmo CHAR. El medio de byte de signo (el primer medio byte del último byte del campo) se convierte de acuerdo con los valores hexadecimales que aparecen en la tabla siguiente.

EBCDIC para el tipo NUM	EBCDIC para el tipo NUMC	ASCII
F (signo positivo)	C	3
D (signo negativo)	D	7

### Consulta relacionada

“Elementos de asociación” en la página 375

“Texto de idioma bidireccional”

“Elemento callLink” en la página 407

“convert()” en la página 897

“callConversionTable” en la página 930

## Texto de idioma bidireccional

Los idiomas bidireccionales (bidi), como por ejemplo el árabe y el hebreo, son idiomas en los que el texto se presenta al usuario ordenado de derecha a izquierda, pero los números y las series alfabéticas latinas del texto se presentan de izquierda a derecha. Además, el orden de aparición de los caracteres dentro de las variables de programa puede variar. El texto se almacena generalmente en orden *lógico*, el orden en que se especifican los caracteres en el campo de entrada.

Estas diferencias en el orden y en otras características asociadas con la presentación requieren que el programa tenga la capacidad de convertir series de texto bidireccional de un formato a otro. Los atributos de conversión bidireccional se especifican en un archivo de tabla de conversión bidireccional (.bct) creado independientemente del programa. El programa hace referencia al nombre de la tabla de conversión para indicar cómo debe realizarse la conversión de atributos.

En todos los casos, la referencia a la tabla de conversión bidireccional se especifica como el nombre de archivo de entre 1 y 8 caracteres sin la extensión .bct. Por ejemplo, si ha creado una tabla de conversión bidireccional denominada mybct.bct, puede establecer el valor de formConversionTable en un programa añadiendo la siguiente sentencia al principio del programa:

```
sysVar.formConversionTable = "mybct.bct" ;
```

Las tareas que debe realizar son las siguientes:

- Cree tablas de conversión bidireccional que especifiquen las transformaciones que deben producirse. Tenga en cuenta que son necesarias tablas diferentes para convertir los datos que deben visualizarse en un formulario de texto o de impresión .
- Al generar un programa que utiliza formularios de texto o de impresión con texto de idioma bidireccional, añada al programa una sentencia que asigne el nombre de tabla de conversión a la función de sistema sysVar.formConversionTable antes de visualizar el formulario.

El archivo de tabla de conversión bidireccional se crea mediante el conector asistente de tabla de conversión bidireccional, que se encuentra en el archivo BidiConversionTable.zip:

1. Baje el archivo del siguiente sitio web:
2. Descomprima el archivo en el directorio del entorno de trabajo

- Para empezar a ejecutar el asistente, pulse **Archivo > Nuevo > Otros > BidiConversionTable**.  
El nombre de una tabla utilizada con programas EGL debe tener ocho caracteres como máximo y la extensión .bct.
- Al ejecutar el asistente, pulse F1 para obtener ayuda en la elección de las opciones correctas para crear la tabla.

### Consulta relacionada

“Conversión de datos” en la página 467

“callConversionTable” en la página 930

---

## Inicialización de datos

Si un programa generado por EGL inicializa un registro automáticamente (como ocurre en algunos casos, que se describen más adelante), cada uno de los elementos de estructura de nivel más bajo se establece en un valor que sea apropiado para el tipo primitivo. La inicialización de formulario es similar, excepto que la declaración de formulario puede asignar valores que alteran temporalmente los valores por omisión.

La inicialización también se produce en las siguientes situaciones:

- La propiedad **initialized** de una variable (concretamente, de un elemento o registro o matriz estática) se establece en *sí*.
- La lógica incluye algunas variaciones de set

La tabla siguiente ofrece detalles sobre los valores de inicialización.

Tipo primitivo	Valor de inicialización
ANY	La variable no es de un tipo definido
BIN y los tipos de entero (BIGINT, INT y SMALLINT), HEX, FLOAT, SMALLFLOAT	Ceros binarios
CHAR, MBCHAR	Blancos de un solo byte
DATE, TIME, TIMESTAMP	Valor actual del reloj del sistema (para el número de bytes necesarios para la máscara en el caso de TIMESTAMP)
DBCHAR	Blancos de doble byte
DECIMAL, MONEY, NUM, NUMC, PACF	Ceros numéricos
INTERVAL	Ceros numéricos (para el número de bytes necesarios para la máscara), precedidos de un signo más
UNICODE	Blancos Unicode (cada uno de los cuales es hexadecimal 0020)

En una estructura, sólo se tienen en cuenta los elementos de estructura de nivel más bajo. Si, por ejemplo, un elemento de estructura de tipo HEX está subordinado a un elemento de estructura de tipo CHAR, el área de memoria se inicializa con ceros binarios.

Los registros o elementos que se reciben como parámetros de programa o función no se inicializan nunca automáticamente.

Un programa generado por EGL inicializa registros, que pueden ser locales o globales.

#### Conceptos relacionados

“Componente de función” en la página 140

“Componente dataItem” en la página 131

“Componente de programa” en la página 139

“Componentes de registro” en la página 132

“Estructura fija” en la página 26

#### Consulta relacionada

“Sentencias EGL” en la página 88

“set” en la página 636

---

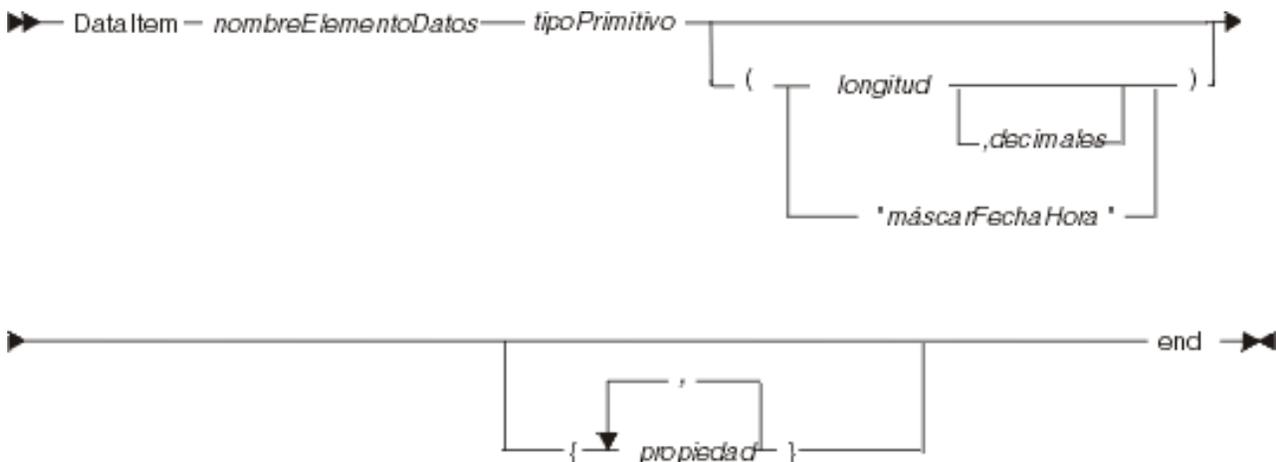
## Componente DataItem en formato fuente EGL

Un componente DataItem se declara en un archivo EGL, como se describe en el apartado *Formato fuente EGL*.

A continuación se ofrece un ejemplo de componente de elemento de datos:

```
DataItem myDataItemPart
 BIN(9,2)
end
```

El diagrama de sintaxis de un componente dataItem es el siguiente:



#### **DataItem** nombreComponenteElementoDatos ... end

Identifica el componente como componente DataItem y especifica el nombre. Para conocer las normas, consulte el apartado *Convenios de denominación*.

#### *tipoPrimitivo*

El tipo primitivo asignado al componente dataItem.

### *longitud*

Entero que refleja la longitud del componente dataItem. El valor de cualquier variable basada en el componente incluye el número especificado de caracteres o dígitos.

### *decimales*

Para un tipo numérico, fijo que no sea MONEY (específicamente BIN, DECIMAL, NUM, NUMC o PACF), puede especificar *decimales*, que es un entero que representa el número de posiciones después de la coma decimal. El número máximo de posiciones decimales es el menor de dos números: 18 o el número de dígitos declarado como *longitud*. La coma decimal no se almacena con los datos.

### *" máscaraFechaHora"*

Para elementos de tipo INTERVAL o TIMESTAMP, puede especificar *máscaraFechaHora*, que asigna un significado (como por ejemplo "dígito de año") a una posición dada en el valor de elemento. La máscara no se almacena con los datos.

### *propiedad*

Una propiedad de elemento, como se describe en el apartado *Visión general de las propiedades EGL y alteraciones temporales*.

## **Conceptos relacionados**

"Componente dataItem" en la página 131

"Proyectos, paquetes y archivos EGL" en la página 13

"Visión general de las propiedades de EGL" en la página 64

"Referencias a componentes" en la página 21

"Componentes" en la página 17

## **Tareas relacionadas**

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

## **Consulta relacionada**

"Formato fuente EGL" en la página 491

"Componente de función en formato fuente EGL" en la página 527

"Componente de registro indexado en formato fuente EGL" en la página 535

"Componente de registro MQ en formato fuente EGL" en la página 662

"Convenios de denominación" en la página 672

"Tipos primitivos" en la página 34

"Componente de programa en formato fuente EGL" en la página 728

"Componente de registro relativo en formato fuente EGL" en la página 740

"Componente de registro serie en formato fuente EGL" en la página 743

"Componente de registro SQL en formato fuente EGL" en la página 748

---

## **Componente DataTable en formato fuente EGL**

Un componente dataTable se declara en un archivo EGL, que está descrito en *Proyectos, paquetes y archivos EGL*. Este componente es un componente generable, lo que significa que debe estar en el nivel superior del archivo y debe tener el mismo nombre que el archivo.

Una dataTable se relaciona con un programa mediante la declaración de uso del programa o, en el caso de la única tabla de mensajes del programa, por la propiedad **msgTablePrefix** del programa. Una dataTable está relacionada con un pageHandler por la declaración de uso del pageHandler.

Este es un ejemplo de un componente dataTable:

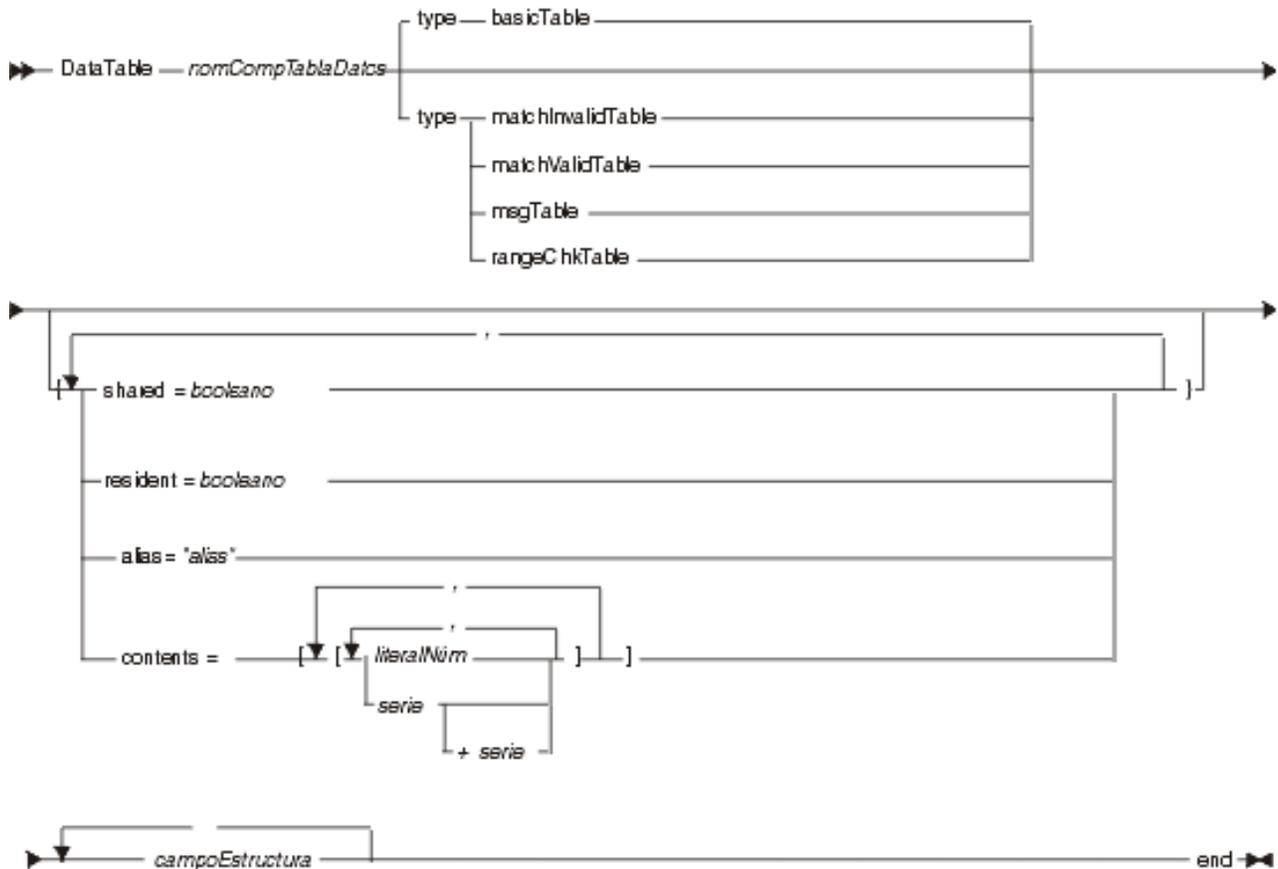
```

DataTable myDataTablePart type basicTable
{
 { shared = yes }
 myColumn1 char(10);
 myColumn2 char(10);
 myColumn3 char(10);

 { contents = [
 ["row1 col1", "row1 col2", "row1 " + "col3"],
 ["row2 col1", "row2 col2", "row2 " + "col3"],
 ["row3 col1", "row3 col2", "row3 col3"]
]
}
end

```

El diagrama de sintaxis para un componente dataTable es el siguiente:



**DataTable nombreComponenteTablaDatos ... end**

Identifica el componente como una dataTable y especifica el nombre del componente. Para conocer las reglas de denominación, consulte *Convenios de denominación*.

**basicTable (el valor por omisión)**

Contiene información que se utiliza en la lógica del programa; por ejemplo, una lista de países y códigos relacionados.

**matchInvalidTable**

Se especifica en la propiedad **validadorDataTable** de un campo de texto para

indicar que la entrada del usuario debe ser distinta a cualquier valor de la primera columna de la dataTable. La ejecución de EGL actúa de la siguiente manera como respuesta a una anomalía de validación:

- Accede a la tabla referenciada en la propiedad **validatorDataTable**
- Recupera el mensaje identificado por la propiedad **validatorDataTableMsgKey** específica del campo de texto
- Visualiza el mensaje en el campo de texto identificado en la propiedad **msgField** específica del formulario

#### **matchValidTable**

Se especifica en la propiedad **validatorDataTable** de un campo de texto para indicar que la entrada del usuario debe coincidir con un valor de la primera columna de la dataTable. La ejecución de EGL actúa de la siguiente manera como respuesta a una anomalía de validación:

- Accede a la tabla referenciada en la propiedad **validatorDataTable**
- Recupera el mensaje identificado por la propiedad **validatorDataTableMsgKey** específica del campo de texto
- Visualiza el mensaje en el campo identificado en la propiedad **msgField** específica del formulario

#### **msgTable**

Contiene mensajes de ejecución. Se presenta un mensaje bajo la siguiente circunstancia:

- La tabla es la tabla de mensajes del programa. La asociación de tabla y programa se produce si la propiedad del programa **msgTablePrefix** hace referencia al *prefijo de tabla*, que es el primero de los cuatro caracteres en el nombre de la dataTable. El resto del nombre es uno de los códigos de idioma nacional de la tabla siguiente.

Idioma	Código de idioma nacional
Portugués de Brasil	PTB
Chino simplificado	CHS
Chino tradicional	CHT
Inglés, mayúsculas	ENP
Inglés, EE.UU.	ENU
Francés	FRA
Alemán	DEU
Italiano	ITA
Japonés, Katakana (juego de caracteres de un solo byte)	JPN
Coreano	KOR
Español	ESP
Alemán de Suiza	DES

- El programa recupera y presenta un mensaje mediante uno de dos mecanismos, tal como se describe en *ConverseLib.displayMsgNum* y *ConverseLib.validationFailed*.

#### **rangeChkTable**

Se especifica en la propiedad **validatorDataTable** de un campo de texto para indicar que la entrada del usuario debe coincidir con un valor se esté entre los valores de la primera y la segunda columnas de, como mínimo, una fila de

tabla de datos. (El rango es inclusivo; la entrada del usuario es válida si coincide con un valor de la primera o segunda columna de cualquier fila.) La ejecución de EGL actúa de la siguiente manera como respuesta a una anomalía de validación:

- Accede a la tabla referenciada en la propiedad **validatorDataTable**
- Recupera el mensaje identificado por la propiedad **validatorDataTableMsgKey** específica del campo de texto
- Visualiza el mensaje en el campo identificado en la propiedad **msgField** específica del formulario

#### " alias"

Una serie incorporada a los nombres de la salida generada. Si no se especifica un alias, en su lugar se utiliza el nombre de dataTable .

#### shared

Indica si múltiples programas de la misma unidad de ejecución utilizan la misma instancia de una dataTable. Los valores válidos son *yes* y *no* (valor por omisión). Si el valor de **shared** es *no*, cada programa de la unidad de ejecución tendrá una copia exclusiva de la dataTable.

La propiedad indica si todos los programas de la misma unidad de ejecución utilizan la misma instancia de una dataTable. Si el valor de **shared** es *no*, cada programa de la unidad de ejecución tendrá una copia exclusiva de la dataTable.

Los cambios realizados durante la ejecución son visibles para cada programa que tenga acceso a la dataTable y los cambios permanecen hasta que se descarga la dataTable. En la mayoría de casos, el valor de la propiedad **resident** (descrito más adelante) determine cuándo se descarga la dataTable; para conocer más detalles, consulte la descripción de esa propiedad.

#### resident

Indica si la dataTable se conserva en la memoria incluso después de finalizar cada programa que ha accedido a la dataTable.

Los valores válidos son *yes* y *no*. El valor predeterminado es *no*.

Si establece la propiedad **resident** en *sí*, la dataTable se comparte independientemente del valor de **shared**.

Las ventajas de que la dataTable sea residente son las siguientes:

- La dataTable retiene los valores grabados en ella por los programas que se ejecutaron anteriormente
- La tabla está disponible para su acceso inmediato sin proceso de carga adicional

Una dataTable residente permanece cargada hasta que la unidad de ejecución finaliza. Una dataTable no residente, no obstante, se descarga cuando el programa que la utiliza finaliza.

**Nota:** Una dataTable se carga en la memoria (si es necesario) en el primer acceso a un programa, no cuando la ejecución de EGL procesa una declaración de uso.

#### content

El valor de las células de la dataTable, cada una de las cuales es de una de las siguientes clases:

- Un literal numérico
- Un literal de serie o una concatenación de literales de serie

La clase de contenido de una fila dada debe ser compatible con los campos de estructura de nivel superior, cada uno de los cuales representa una definición de columna.

*campoEstructura*

Un campo de estructura, como se describe en la sección *Campo de estructura en formato fuente EGL*.

**Conceptos relacionados**

“DataTable” en la página 146

“Proyectos, paquetes y archivos EGL” en la página 13

“Unidad de ejecución” en la página 742

**Consulta relacionada**

“Convenios de denominación” en la página 672

“Elemento de estructura en el formato fuente de EGL” en la página 752

“displayMsgNum()” en la página 789

“validationFailed()” en la página 791

“Declaración use” en la página 954

---

## Vía de acceso de construcción EGL y eglpath

Todo proyecto EGL y Web EGL se asocia con una vía de acceso de construcción EGL a fin de que el proyecto pueda hacer referencia a los componentes de otros proyectos. Para obtener detalles acerca de cuándo se utiliza la vía de acceso de construcción EGL y por qué es importante el orden de las entradas de la misma, consulte el apartado *Referencias a componentes*.

Al especificar la vía de acceso de construcción EGL, puede elegir *exportar* uno o varios de los proyectos que se encuentran en la misma. A continuación, cuando un proyecto haga referencia al proyecto que se declara, cada uno de los proyectos exportados quedará a disposición del proyecto referenciador, como en el ejemplo siguiente:

- La vía de acceso de construcción EGL del proyecto A contiene por orden los siguientes proyectos:

A, B, C, D

Se exportan los proyectos B y D.

- La vía de acceso de construcción EGL del proyecto L contiene por orden los siguientes proyectos:

L, J, A, Z

- La vía de acceso de construcción real del proyecto L también incluye los proyectos exportados del proyecto A. En este caso, la vía de acceso de construcción EGL del proyecto L es en realidad la siguiente:

L, J, A, B, D, Z

Los proyectos exportados se colocan a continuación del proyecto que los exporta, en el orden en el que aparecen en la vía de acceso de construcción del proyecto exportador.

La vía de acceso de construcción de un proyecto incluye siempre el propio proyecto, que generalmente es el primero del orden de la vía de acceso de construcción, lo cual es aconsejable. Si tiene varias carpetas fuente EGL en el proyecto, todas ellas deben aparecer en la vía de acceso de construcción EGL de esa proyecto, y cualquier proyecto que haga referencia al proyecto utilizará el orden de dichas carpetas.

*Es muy aconsejable evitar que existan paquetes con nombres idénticos en proyectos o carpetas diferentes del mismo proyecto.*

Si genera en el SDK de EGL, la situación es la siguiente:

- No hay información disponible sobre el proyecto.
- El argumento de línea de mandatos *eglpath* sustituye a la función de la vía de acceso de construcción de EGL. *eglpath* es una lista de directorios de sistema operativo en los que se busca cuando el SDK de EGL intenta resolver una referencia de componente.
- Las normas de utilización de *eglpath* son equivalentes a las de utilización de la vía de acceso de construcción EGL; sin embargo, no puede exportar directorios como exporta los proyectos.

*Al utilizar el SDK de EGL, es muy aconsejable evitar que existan paquetes con nombres idénticos en directorios diferentes.*

#### **Conceptos relacionados**

“Generación a partir del SDK (Software Development Kit) de EGL” en la página 333  
“Referencias a componentes” en la página 21

#### **Tareas relacionadas**

“Generar a partir del SDK (Software Development Kit) de EGL” en la página 333

#### **Consulta relacionada**

“EGLSDK” en la página 488

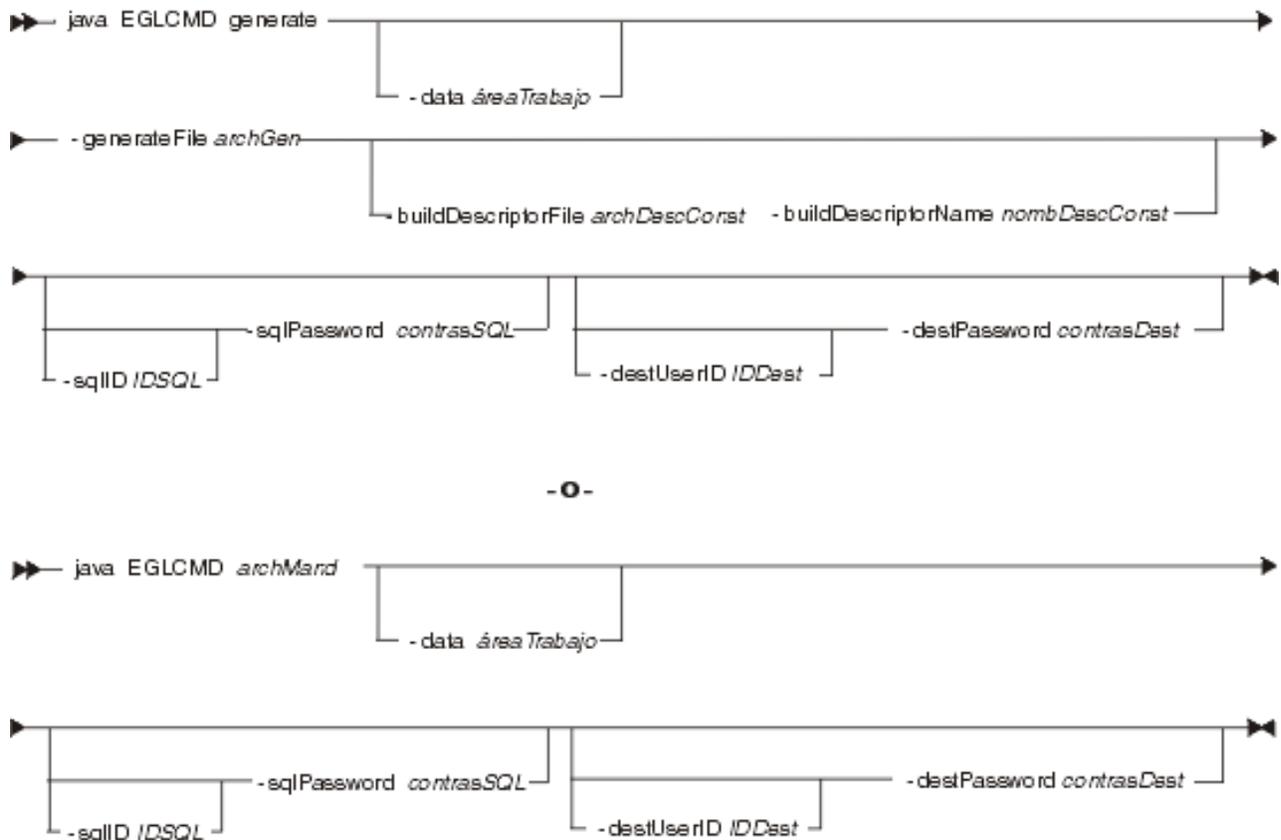
---

## **EGLCMD**

El mandato EGLCMD le otorga acceso a la interfaz de proceso por lotes del entorno de trabajo, como se describe en *Generación desde la interfaz de proceso por lotes del entorno de trabajo*.

### **Sintaxis**

La sintaxis para invocar a EGLCMD es la siguiente:



- O -

### generate

Indica que el propio mandato hace referencia al archivo EGL y al componente descriptor de construcción que se utilizan para generar salida. En este caso, el mandato EGLCMD no hace referencia a un archivo de mandatos.

### -data *áreaTrabajo*

Especifica la vía de acceso absoluta o relativa del directorio del área de trabajo. Las vías de acceso relativas son relativas al directorio en el que ejecuta el mandato.

Si no especifica un valor, el mandato accede al entorno de trabajo predeterminado de Eclipse.

Escriba la vía de acceso entre comillas.

### *archivoMdt*

Especifica la vía de acceso absoluta o relativa del archivo descrito en *archivo de mandatos EGL*. Las vías de acceso relativas son relativas al directorio en el que ejecuta el mandato.

Escriba la vía de acceso entre comillas.

El archivo de mandatos debe estar en el espacio de trabajo; en caso contrario, utilice el proceso de importación de Eclipse para importar el archivo y luego vuelva a ejecutar EGLCMD.

### -generateFile *archivoGen*

Especifica la vía de acceso absoluta o relativa del archivo EGL que contiene el componente que desea procesar. Las vías de acceso relativas son relativas al directorio en el que ejecuta el mandato.

Escriba la vía de acceso entre comillas.

**-buildDescriptorFile** *archivodc*

Especifica la vía de acceso absoluta o relativa del archivo de construcción que contiene el descriptor de construcción. Las vías de acceso relativas son relativas al directorio en el que ejecuta el mandato.

Escriba la vía de acceso entre comillas.

**-buildDescriptorName** *nombredc*

Especifica el nombre de un componente descriptor de construcción que guía la generación. El descriptor de construcción debe estar en el nivel superior de un archivo de construcción de EGL (.eglbld).

**-sqlID** *IDSql*

Establece el valor de la opción de descriptor de construcción sqlID.

**-sqlPassword** *conSQL*

Establece el valor de la opción de descriptor de construcción sqlPassword.

**-destUserid** *IDdest*

Establece el valor de la opción de descriptor de construcción destUserID.

**-destPassword** *contDest*

Establece el valor de la opción de descriptor de construcción destPassword.

Las opciones del descriptor de construcción que especifique al invocar al mandato EGLCMD tienen prioridad sobre las opciones del descriptor de construcción (si las hay) que están listadas en el archivo de mandatos de EGL.

## Ejemplos

En los mandatos indicados a continuación, cada ejemplo de varias líneas debe estar en una sola línea:

```
java EGLCMD "commandfile.xml"
```

```
java EGLCMD "commandfile.xml" -data "c:\myWorkSpace"
```

```
java EGLCMD generate
-generateFile "c:\myProg.eglpgm"
-data "myWorkSpace"
-buildDescriptorFile "c:\myBuild.eglbld"
-buildDescriptorName myBuildDescriptor
```

```
java EGLCMD "myCommand.xml"
-data "my WorkSpace"
-sqlID myID -sqlPassword myPW
-destUserID myUserID -destPassword myPass
```

### Conceptos relacionados

“Generación a partir de la interfaz por lotes del entorno de trabajo” en la página 332

### Tareas relacionadas

“Generar desde la interfaz por lotes del entorno de trabajo” en la página 331

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### Consulta relacionada

“destPassword” en la página 389

“destUserID” en la página 390

“sqlID” en la página 401

“sqlPassword” en la página 403

## Archivo de mandatos EGL

Un archivo de mandatos EGL indica qué archivos EGL desea procesar al generar salida fuera del entorno de trabajo, ya esté utilizando la interfaz de proceso por lotes del entorno de trabajo (mandato EGLCMD) o el EGL SDK (mandato EGLSDK). Puede crear el archivo de una de dos maneras:

- Manualmente, según las normas descritas más adelante, o bien
- Utilizando el asistente para la Generación de EGL, como se describe en *Generación en el entorno de trabajo*.

El archivo de mandatos es un archivo XML y el nombre de archivo debe tener la extensión .xml, en cualquier combinación de letras mayúsculas y minúsculas. El contenido del archivo debe estar en conformidad con la siguiente definición de tipo de documento (DTD):

```
dirInstalación\egl\eclipse\plugins\
com.ibm.etools.egl.utilities_versión\
dtd\eglcommands_5_1.dtd
```

### *dirInstalación*

El directorio de instalación del producto, como por ejemplo C:\Program Files\IBM\RSPD\6.0. Si instaló y tuvo un producto de Rational Developer antes de instalar el producto que está utilizando ahora, deberá especificar el directorio utilizado en la instalación anterior.

### *versión*

La versión instalada del conector; por ejemplo, 6.0.0

La tabla siguiente muestra los elementos y atributos soportados por la DTD. Los nombres de elemento y atributo son sensibles a las mayúsculas y minúsculas.

Elemento	Atributo	Valor de atributo
EGLCOMMANDS (necesario)	eglpath	<p>Tal como se describe en <i>eglpath</i>, el atributo eglpath identifica directorios en los que buscar cuando EGL utiliza una sentencia de importar para resolver el nombre de un componente. El atributo es opcional y, si está presente, hace referencia a una serie entrecomillada que tiene uno o varios nombres de directorio, cada uno separado del siguiente por punto y coma.</p> <p>El atributo se utiliza solamente si el mandato EGLSDK hace referencia al archivo de mandatos. Si el mandato EGLCMD está utilizándose, se ignora el valor de eglpath; en su lugar, las sentencias de importar se resuelven de acuerdo con la vía de acceso del proyecto EGL, tal como se describe en <i>Importar</i>.</p>

Elemento	Atributo	Valor de atributo
<b>buildDescriptor</b> (opcional; puede evitar especificar este valor si está utilizando un descriptor de construcción maestro, como se describe en <i>Componente descriptor de construcción</i> )	<b>nombre</b>	El nombre de un componente descriptor de construcción que guía la generación. El descriptor de construcción debe estar en el nivel superior de un archivo de construcción de EGL (.eglbld).  Las opciones del descriptor de construcción que especifique al invocar a EGLCMD o EGLSDK tienen prioridad sobre las opciones del descriptor de construcción (si las hay) que están listadas en el archivo de mandatos de EGL.
	<b>archivo</b>	La vía de acceso absoluta o relativa del archivo EGL que contiene el descriptor de construcción. Las vías de acceso relativas especificadas para EGLCMD son relativas al nombre de vía de acceso del espacio de trabajo de Enterprise Developer. Las vías de acceso relativas especificadas para EGLSDK son relativas al directorio en el que ejecuta el mandato.  La vía de acceso debe estar entre comillas si la vía de acceso incluye un espacio.
<b>generar</b> (opcional)	<b>archivo</b>	La vía de acceso absoluta o relativa del archivo EGL que contiene el componente que desea procesar. Las vías de acceso relativas especificadas para EGLCMD son relativas al nombre de vía de acceso del espacio de trabajo de Enterprise Developer. Las vías de acceso relativas especificadas para EGLSDK son relativas al directorio en el que ejecuta el mandato.  La vía de acceso debe estar entre comillas si la vía de acceso incluye un espacio.  Si omite el atributo de archivo, no se produce generación.

## Ejemplos de archivos de mandatos

Esta sección muestra dos archivos de mandatos. Los resultados generados por cualquiera de los dos archivos son los mismos ya utilice el mandato EGLCMD o el mandato EGLSDK, si ejecuta el mandato EGLSDK en el directorio en el que residen los archivos de programa de EGL.

El siguiente archivo de mandatos contiene un mandato generate que utiliza el descriptor de construcción myBDescPart para generar el programa myProgram.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE EGLCOMMANDS PUBLIC "-//IBM//DTD EGLCOMMANDS 5.1//EN" "">
<EGLCOMMANDS eglpath="C:\mydata\entdev\workspace\projectinteract">
 <generate file="projectinteract\myProgram.eglpgm">
 <buildDescriptor name="myBDescPart" file="projectinteract\mybdesc.eglbld"/>
 </generate>
</EGLCOMMANDS>
```

El siguiente ejemplo contiene dos mandatos generate y ambos utilizan de forma implícita un descriptor de construcción maestro.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE EGLCOMMANDS PUBLIC "-//IBM//DTD EGLCOMMANDS 5.1//EN" "">
<EGLCOMMANDS eglpath="C:\mydata\entdev\workspace\projecttrade">
 <generate file="projecttrade\program2.eglpgm"/>
 <generate file="projecttrade\program3.eglpgm"/>
</EGLCOMMANDS>
```

### Conceptos relacionados

“Componente descriptor de construcción” en la página 293  
“Generación a partir del SDK (Software Development Kit) de EGL” en la página 333  
“Generación a partir de la interfaz por lotes del entorno de trabajo” en la página 332  
“Import” en la página 33

### Tareas relacionadas

“Generar a partir del SDK (Software Development Kit) de EGL” en la página 333  
“Generar desde la interfaz por lotes del entorno de trabajo” en la página 331  
“Generar en el entorno de trabajo” en la página 329

### Consulta relacionada

“EGLCMD” en la página 478  
“Archivo de mandatos EGL” en la página 481  
“Vía de acceso de construcción EGL y eglpath” en la página 477  
“EGLSDK” en la página 488

---

## Editor EGL

Para modificar un archivo EGL (extensión .egl), trabaje en el editor de fuente EGL, que le guiará con ayuda de contenido.

Para modificar un archivo de construcción de EGL (extensión .eglbld), siga uno de estos procedimientos:

- Crear un archivo de construcción
- 
- Añadir un componente de descriptor de construcción
- Añadir un componente de opciones de enlace
- 
- Añadir un componente de asociaciones de recursos

### Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13  
“Componentes” en la página 17

### Consulta relacionada

“Ayuda de contenido en EGL”

## Ayuda de contenido en EGL

El editor EGL proporciona *ayuda de contenido*, que propone información que puede añadirse al archivo fuente. Con una o dos pulsaciones de teclas, el usuario puede completar el nombre de un componente, variable o función o colocar una *plantilla* (el esquema de un componente) en el archivo fuente.

La pulsación que activa la ayuda de contenido es **Control + Espacio**.

### Tareas relacionadas

“Establecer preferencias para plantillas” en la página 118

“Utilizar las plantillas EGL con la ayuda de contenido” en la página 129

---

## Enumeraciones en EGL

En algunos casos en EGL, los valores de una propiedad o campo se restringen a los valores de una determinada *enumeración*, que es una categoría de valores predefinidos. Por ejemplo, la propiedad **color** acepta un valor de la enumeración **ColorKind** y los valores válidos de dicha enumeración son *white* y *red*.

Puede calificar un valor de enumeración con el nombre de enumeración, de modo que los valores precedentes pueden indicarse como *ColorKind.white* y *ColorKind.red*. Sin embargo, sólo debe calificar el valor de enumeración cuando el código tiene acceso a una variable o constante cuyo nombre es el mismo que el valor de enumeración. Si, por ejemplo, una variable llamada *red* está en el ámbito, el símbolo *red* hace referencia a la variable y no al valor de enumeración.

La lista siguiente de enumeraciones incluye los valores de enumeración; sin embargo, las explicaciones de dichos valores aparecen en otro lugar, en el contexto de la propiedad o valor donde la enumeración es significativa:

### **AlignKind**

- center
- left
- none
- right

### **Boolean**

- yes
- no

### **CallingConventionKind**

- I4GL
- Library

### **CaseFormatKind**

- defaultCase
- lower
- upper

### **ColorKind**

- black (ya que sólo es válido para campos de consola)
- blue
- cyan
- defaultColor
- green
- magenta
- red
- yellow
- white

### **DataSource**

- databaseConnection

reportData  
sqlStatement

**DeviceTypeKind**

doubleByte  
singleByte

**DisplayUseKind**

button  
hyperlink  
input  
output  
secret  
table

**EventKind**

AFTER\_DELETE  
AFTER\_FIELD  
AFTER\_OPENUI  
AFTER\_INSERT  
AFTER\_ROW  
BEFORE\_DELETE  
BEFORE\_FIELD  
BEFORE\_OPENUI  
BEFORE\_INSERT  
BEFORE\_ROW  
ON\_KEY  
MENU\_ACTION

**ExportFormat**

html  
pdf  
text  
xml

**HighlightKind**

blink  
defaultHighlight  
noHighlight  
reverse  
underline

**IndexOrientationKind**

across  
down

**IntensityKind**

bold  
defaultHighlight  
dim

invisible  
normalIntensity

#### **LineWrapKind**

character  
compress (ya que sólo es válido para campos de consola)  
word

#### **OutlineKind**

bottom  
left  
right  
top

**Nota:** `sysLib.box` es una constante equivalente a `[left,right,top,bottom]`.  
`sysLib.noOutline` es una constante que significa que no hay contorno.

#### **PfKeyKind**

`pf $n$` , donde  $(1 \leq n \leq 24)$

#### **ProtectKind**

skip  
no  
yes

#### **SelectTypeKind**

index  
value

#### **SignKind**

leading  
none  
parens  
trailing

#### **Conceptos relacionados**

“Visión general de las propiedades de EGL” en la página 64

“Referencias a variables en EGL” en la página 58

---

## **Palabras reservadas EGL**

EGL incluye dos categorías de palabras reservadas:

- Palabras reservadas para usos específicos excepto al trabajar en una sentencia SQL
- Palabras reservadas para usos específicos al trabajar en una sentencia SQL

## **Palabras reservadas fuera de una sentencia SQL**

Fuera de las sentencias SQL, las palabras reservadas son las siguientes en cualquier combinación de mayúsculas y minúsculas:

- absolute, add, all, any, as
- bigInt, bin, bind, blob, boolean, by, byName, byPosition
- call, case, char, clob, close, const, continue, converse, current
- dataItem, dataTable, date, dbChar, decimal, decrement, delete, display, dliCall

- else, embed, end, escape, execute, exit, externallyDefined
- false, field, first, float, for, forEach, form, formGroup, forUpdate, forward, freeSql, from, function
- get, goto
- handler, hex, hold
- if, import, in, inOut, insert, int, interval, into, is, isa
- label, languageBundle, last, library, like
- matches, mbChar, money, move
- new, next, no, noRefresh, not, nullable, num, number, numc
- onEvent, onException, open, openUI, otherwise, out
- pacf, package, pageHandler, passing, prepare, previous, print, private, program, psb
- record, ref, relative, replace, return, returning, returns
- scroll, self, set, show, singleRow, smallFloat, smallInt, sql, sqlCondition, stack, string
- this, time, timeStamp, to, transaction, transfer, true, try, type
- unicode, update, url, use, using, usingKeys
- when, while, with, withinParent
- yes

### **Palabras reservadas en una sentencia SQL**

En las sentencias SQL, las palabras reservadas son las siguientes en cualquier combinación de mayúsculas y minúsculas:

- absolute, action, add, alias, all, allocate, alter, and, any, are, as, asc, assertion, at, authorization, avg
- begin, between, bigint, binaryLargeObject, bit, bit\_length, blob, boolean, both, by
- call, cascade, cascaded, case, cast, catalog, char, char\_length, character, character\_length, characterLargeObject, characterVarying, charLargeObject, charVarying, check, clob, close, coalesce, collate, collation, column, comment, commit, connect, connection, constraint, constraints, continue, convert, copy, corresponding, count, create, cross, current, current\_date, current\_time, current\_timestamp, current\_user, cursor
- data, database, date, dateTime, day, deallocate, dec, decimal, declare, default, deferrable, deferred, delete, desc, describe, diagnostics, disconnect, distinct, domain, double, doublePrecision, drop
- else, end, endExec, escape, except, exception, exec, execute, exists, explain, external, extract
- false, fetch, first, float, for, foreign, found, from, full
- get, getCurrentConnection, global, go, goto, grant, group
- having, hour
- identity, image, immediate, in, index, indicator, initially, inner, input, insensitive, insert, int, integer, intersect, into, is, isolation
- join
- key
- language, last, leading, left, level, like, local, long, longint, lower, ltrim
- match, max, min, minute, module, month
- national, nationalCharacter, nationalCharacterLargeObject, nationalCharacterVarying, nationalCharLargeObject, nationalCharVarying, natural, nchar, ncharVarying, nclob, next, no, not, null, nullIf, number, numeric
- octet\_length, of, on, only, open, option, or, order, outer, output, overlaps
- pad, partial, position, prepare, preserve, primary, prior, privileges, procedure, public
- raw, read, real, references, relative, restrict, revoke, right, rollback, rows, rtrim, runtimeStatistics



descriptor de construcción que se utilizan para generar salida. En este caso, el mandato EGLSDK no hace referencia a un archivo de mandatos.

*archivoMdt*

Especifica la vía de acceso absoluta o relativa del archivo descrito en el apartado *Archivo de mandatos EGL*. Las vías de acceso relativas son relativas al directorio en el que ejecuta el mandato.

Escriba la vía de acceso entre comillas.

**-eglpath** *eglpath*

Tal como se describe en *eglpath*, la opción *eglpath* identifica directorios en los que buscar cuando EGL utiliza una sentencia de importar para resolver el nombre de un componente. Especificará una serie entrecomillada que tenga uno o varios nombres de directorio, cada uno separado del siguiente por punto y coma.

**-generateFile** *archivoGen*

La vía de acceso absoluta o relativa del archivo EGL que contiene el componente que desea procesar. Las vías de acceso relativas son relativas al directorio en el que ejecuta el mandato.

Escriba la vía de acceso entre comillas.

**-buildDescriptorFile** *archivodc*

La vía de acceso absoluta o relativa del archivo de construcción que contiene el descriptor de construcción. Las vías de acceso relativas son relativas al directorio en el que ejecuta el mandato.

Escriba la vía de acceso entre comillas.

**-buildDescriptorName** *nombredc*

El nombre de un componente descriptor de construcción que guía la generación. El descriptor de construcción debe estar en el nivel superior de un archivo de construcción de EGL (.eglbld).

**-sqlID** *IDSql*

Establece el valor de la opción de descriptor de construcción *sqlID*.

**-sqlPassword** *conSQL*

Establece el valor de la opción de descriptor de construcción *sqlPassword*.

**-destUserid** *IDdest*

Establece el valor de la opción de descriptor de construcción *destUserID*.

**-destPassword** *contDest*

Establece el valor de la opción de descriptor de construcción *destPassword*.

El valor de *eglpath* que especifique al invocar al mandato EGLSDK tiene prioridad sobre cualquier valor de *eglpath* en un archivo de mandatos EGL. De forma similar, las opciones del descriptor de construcción que especifique al invocar al mandato tienen prioridad sobre las opciones de cualquier descriptor de construcción que están listadas en un archivo de mandatos de EGL.

## Ejemplos

En los mandatos indicados a continuación, cada ejemplo de varias líneas debe estar en una sola línea:

```
java EGLSDK "commandfile.xml"

java EGLSDK "commandfile.xml"
 -eglpath "c:\myGroup;h:\myCorp"
```

```

java EGLSDK generate
 -eglpath "c:\myGroup;h:\myCorp"
 -generateFile "c:\myProg.eglpgm"
 -buildDescriptorFile "c:\myBuild.eglbld"
 -buildDescriptorName myBuildDescriptor

java EGLSDK "myCommand.xml"
 -sqlID myID -sqlPassword myPW
 -destUserID myUserID -destPassword myPass

```

### Conceptos relacionados

"Componente descriptor de construcción" en la página 293  
 "Generación a partir del SDK (Software Development Kit) de EGL" en la página 333  
 "Import" en la página 33  
 "Descriptor de construcción maestro" en la página 297

### Tareas relacionadas

"Generar a partir del SDK (Software Development Kit) de EGL" en la página 333

### Consulta relacionada

"destPassword" en la página 389  
 "destUserID" en la página 390  
 "Vía de acceso de construcción EGL y eglpath" en la página 477  
 "sqlID" en la página 401  
 "sqlPassword" en la página 403  
 "Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

## Formato del archivo eglmaster.properties

El archivo eglmaster.properties es un archivo de propiedades Java que el SDK de EGL utiliza para especificar el nombre y el nombre de vía de acceso a archivo del descriptor de construcción maestro. Este archivo de propiedades debe estar contenido en un directorio que se especifique en la variable CLASSPATH del proceso que invoca al mandato EGLSDK. El formato del archivo eglmaster.properties es el siguiente:

```

masterBuildDescriptorName=desc
masterBuildDescriptorFile=víaacceso

```

donde:

*desc*

El nombre del descriptor de construcción maestro

*víaacceso*

El nombre de vía de acceso totalmente calificado del archivo EGL en el que se declara el descriptor de construcción maestro utilizado por el EGL SDK

El contenido de este archivo debe seguir las normas de un archivo de propiedades Java. Puede utilizar una barra inclinada (/) o dos barras inclinadas invertidas (\) para separar nombres de archivo dentro de un nombre de vía de acceso.

Debe especificar las palabras clave **masterBuildDescriptorName** y **masterBuildDescriptorFile** en el archivo de propiedades. De lo contrario, se ignorará el archivo eglmaster.properties.

A continuación se muestra un ejemplo del contenido de un archivo eglmaster.properties:

```
Especifique el nombre del descriptor de construcción maestro:
masterBuildDescriptorName=MYBUILDDSCRIPTOR
Especifique el archivo que contiene el descriptor de construcción maestro:
masterBuildDescriptorFile=d:/egl/builddescriptors/master.egl
```

### Conceptos relacionados

“Descriptor de construcción maestro” en la página 297

### Tareas relacionadas

“Elegir opciones para la generación de Java” en la página 300

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

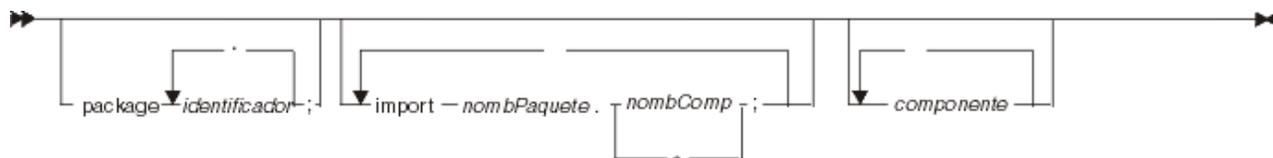
“EGLSDK” en la página 488

“Formato del archivo plugin.xml del descriptor de construcción maestro” en la página 506

---

## Formato fuente EGL

Los componentes lógicos, de datos y de interfaz de usuario se declaran en archivos fuente de EGL, cada uno de los cuales tiene la extensión *.egl* y se construye de la forma siguiente:



### **package** *identificador*

Especifica el nombre del paquete en el que reside el archivo, con cada identificador separado del texto por un punto.

Para obtener una visión general, consulte el apartado *Proyectos, paquetes y archivos EGL*.

### **import** *nombrePaquete*

Especifica el nombre completo de un paquete que debe importarse. Para obtener una visión general, consulte el apartado *Importar*.

### *nombreComponente*

Especifica un solo componente que debe importarse.

\* Indica que todos los componentes del paquete deben importarse.

### *componente*

Uno de los componentes lógicos, de datos de interfaz de usuario de EGL.

Puede especificar comentarios en un archivo EGL, dentro o fuera de los componentes.

### Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13

“Import” en la página 33

“Referencias a componentes” en la página 21

“Componentes” en la página 17

### Tareas relacionadas

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

### Consulta relacionada

"Componente de registro básico en formato fuente EGL" en la página 379

"Comentarios" en la página 438

"Componente DataItem en formato fuente EGL" en la página 472

"Componente DataTable en formato fuente EGL" en la página 473

"Componente FormGroup en formato fuente EGL" en la página 508

"Componente de formulario en formato fuente EGL" en la página 511

"Componente de función en formato fuente EGL" en la página 527

"Componente de registro indexado en formato fuente EGL" en la página 535

"Componente de biblioteca en formato fuente EGL" en la página 649

"Componente de registro MQ en formato fuente EGL" en la página 662

"Componente PageHandler en formato fuente EGL" en la página 679

"Componente de programa en formato fuente EGL" en la página 728

"Componente de registro relativo en formato fuente EGL" en la página 740

"Componente de registro serie en formato fuente EGL" en la página 743

"Componente de registro SQL en formato fuente EGL" en la página 748

---

## Excepciones del sistema de EGL

Las excepciones del sistema de EGL están disponibles en todo código, pero se utilizan con más frecuencia en un bloque **onException**. Para obtener una visión general, consulte la sección *Manejo de excepciones*.

Cada una de las excepciones del sistema de EGL tiene como mínimo los campos siguientes:

### code

Una serie que identifica la excepción, por ejemplo  
"com.ibm.egl.InvocationException" o la constante equivalente  
SysLib.InvocationException

### description

Una serie que indica el significado de la excepción

Las excepciones del sistema EGL son las siguientes:

### SysLib.FileIOException

Identifica un error que se produce durante el acceso a archivos. Los errores que se producen durante el acceso a colas de mensajes de bases de datos relacionales no provocan esta excepción. Los campos específicos son los siguientes:

#### errorCode

El código de estado de 8 caracteres devuelto también en SysVar.ErrorCode; para conocer los detalles, consulte la sección *SysVar.ErrorCode*

#### fileName

El nombre lógico del archivo al que se accede, para conocer los detalles, consulte la sección *Asociaciones de recursos y tipos de archivo*

### SysLib.InvocationException

Identifica un error que se produce en una sentencia **call**.

Los campos específicos son los siguientes:

**errorCode**

El código de estado de 8 caracteres devuelto también en `SysVar.ErrorCode`; para conocer los detalles, consulte la sección *SysVar.ErrorCode*

**name**

El nombre del programa que se llama.

**SysLib.LobProcessingException**

Identifica un error que se ha producido durante el proceso de un campo de tipo LOB o CLOB. Los campos específicos son los siguientes:

**itemName**

Nombre del campo

**operation**

Nombre de la función del sistema EGL que ha fallado

**resource**

Nombre del archivo (si lo hay) conectado al campo

**SysLib.MQIOException**

Identifica un error que se produce durante el acceso de una cola de mensajes de MQSeries. Los campos específicos son los siguientes:

**errorCode**

El código de estado de 8 caracteres devuelto también en `SysVar.ErrorCode`; para conocer los detalles, consulte la sección *SysVar.ErrorCode*

**mqConditionCode**

El código de finalización de una llamada API de MQSeries, tal como se describe en *VGVar.mqConditionCode*

**name**

El nombre lógico de la cola a la que se accede, para conocer los detalles, consulte la sección *Asociaciones de recursos y tipos de archivo*

**SysLib.SQLException**

Identifica un error que se produce durante el acceso de una base de datos relacional. Los campos específicos son los siguientes:

**sqlca**

El área de comunicaciones de SQL; para conocer los detalles, consulte la sección *SysVar.sqlca*

**sqlcode**

El código de retorno de SQL; para conocer los detalles, consulte la sección *SysVar.sqlcode*

**sqlErrd**

Una matriz de 6 elementos, en la que cada elemento contiene el valor de área de comunicación SQL (SQLCA) correspondiente devuelto desde la última operación de E/S SQL; para conocer los detalles, consulte la sección *VGVar.sqlErrd*

**sqlErrmc**

El mensaje de error asociado a `sqlcode`, para un acceso a base de datos que no sea a través de JDBC; para conocer los detalles, consulte la sección *VGVar.sqlErrmc*

**sqlState**

El valor de estado de SQL para la operación de E/S de SQL finalizada más recientemente; para conocer los detalles, consulte la sección *SysVar.sqlState*

### sqlWarn

Una matriz de 11 elementos, en la que cada elemento contiene un byte de aviso devuelto en el área de comunicaciones SQL (SQLCA) para la última operación de E/S SQL y en la que el índice es superior en uno al número de aviso de la descripción SQLCA SQL; para conocer los detalles consulte la sección *VGVar.sqlState*

### Conceptos relacionados

“Asociaciones de recursos y tipos de archivo” en la página 304

### Consulta relacionada

“Excepciones del sistema de EGL” en la página 492

“errorCode” en la página 931

“sqlca” en la página 935

“sqlcode” en la página 936

“sqlState” en la página 936

“mqConditionCode” en la página 948

“sqlerrd” en la página 949

“sqlerrmc” en la página 950

“sqlWarn” en la página 951

---

## Límites de sistema EGL

No está en vigor ningún límite definido por EGL para el número de componentes o el número de niveles jerárquicos de un archivo EGL. Sin embargo, se aplican los siguientes límites:

- Un programa no puede utilizar más de 32767 variables y literales, incluyendo campos en variables.
- Una sentencia call no puede tener más de 30 argumentos; además, se aplican las siguientes restricciones respecto al tamaño de los argumentos en total:
  - No puede haber más de 32567 si remoteCall o.ejbCall es el valor de la propiedad **type** de la llamada.Ambas propiedades están en el componente de opciones de enlace, elemento callLink.
- Un campo no puede tener más de 32767 bytes.
- En la mayoría de los casos, un campo o literal numérico no puede tener más de 32 dígitos más un signo, coma decimal o ambos; pero un campo que recibe el resultado creado invocando la función **mathLib.round** no puede tener más de 31 dígitos más un signo, una coma decimal o ambos.
- Una matriz estática no puede tener más de 7 dimensiones ni más de 32767 en total.
- En el caso de una matriz dinámica, la situación es la siguiente:
  - Una matriz dinámica no puede tener más de 14 dimensiones. El número de dimensiones de una matriz estática de registros es una (para la declaración de registro de matriz), más el número de dimensiones de la estructura de registros.
  - Una matriz dinámica puede tener un tamaño máximo no superior a 2.147.483.647 elementos. Ese número estará en vigor si no especifica un tamaño máximo, pero el tamaño que puede asignarse queda aún más limitado por la memoria disponible durante la ejecución.
  - El tamaño total de todos los argumentos que pueden pasarse en una llamada remota está limitado por el tamaño máximo del almacenamiento intermedio del protocolo.

### Consulta relacionada

"Elemento callLink" en la página 407

"round()" en la página 854

"Convenios de denominación" en la página 672

"Propiedad parmForm del elemento callLink" en la página 416

"Propiedad type del elemento callLink" en la página 423

---

## Expresiones

Una expresión es una serie de operandos y operadores que se especifican al escribir un script de función o programa.

Cada expresión se resuelve en un tipo de valor determinado durante la ejecución. Una *expresión numérica* se resuelve en un número, una *expresión de serie* se resuelve en una serie de caracteres, una *expresión lógica* se resuelve en verdadero o falso y una *expresión de fecha y hora* se resuelve en una fecha, un intervalo, una hora o una indicación de la hora.

Las expresiones se evalúan de acuerdo con un conjunto de reglas de preferencia y (dentro de un nivel de preferencia dado) de izquierda a derecha, pero puede utilizar paréntesis para forzar un orden distinto. Una subexpresión entre paréntesis, anidada se evalúa antes que la subexpresión entre paréntesis encerradora y todas las expresiones entre paréntesis se evalúan antes que la expresión en su totalidad.

En un nivel de evaluación dado, el primer operando determina el tipo de expresión (o subexpresión). Considere el siguiente ejemplo:

```
"Un valor = " + 1 + 2
```

El primer operando es de tipo carácter y la expresión es una expresión de texto con el valor siguiente:

```
"Un valor = 12"
```

Observe una expresión de texto distinta:

```
"un valor = " + (1 + 2)
```

El valor en este caso es el siguiente:

```
"Un valor = 3"
```

### Consulta relacionada

"Expresiones de fecha y hora"

"Expresiones lógicas" en la página 497

"Expresiones numéricas" en la página 504

"Expresiones de texto" en la página 505

## Expresiones de fecha y hora

Una *expresión de fecha y hora* se resuelve en un valor de tipo DATE, INT, INTERVAL, TIME o TIMESTAMP, dependiendo del contexto. Una expresión de fecha y hora debe incluir uno de los elementos siguientes:

- Una variable que contenga un valor de uno de estos tipos.
- Una invocación de función que devuelva un valor de fecha y hora. Varias funciones del sistema crean un valor de fecha y hora a partir de un literal de serie o una constante:
  - **DateTimeLib.dateValue** crea una fecha

- **DateTimeLib.intervalValue** crea un intervalo
- **DateTimeLib.timeValue** crea una hora
- **DateTimeLib.timeStampValue** crea una indicación de la hora

Además, la función del sistema **DateTimeLib.extend** devuelve un valor de indicación de la hora más largo o más corto que un campo de entrada de tipo DATE, TIME o TIMESTAMP.

La tabla siguiente resume los tipos de operaciones aritméticas válidas en una expresión de fecha y hora. Tal como se muestra, una expresión de fecha y hora puede incluir una expresión numérica que devuelva un número, pero sólo en un subconjunto de casos.

Operaciones aritméticas en una expresión de fecha y hora

Tipo de operando 1	Operador	Tipo de operando 2	Tipo de resultado	Comentarios
DATE	-	DATE	INT	
DATE	+/-	NUMBER	DATE	
NUMBER	+	DATE	DATE	
TIME STAMP	-	TIMESTAMP	INTERVAL	INTERVAL(dd, ss) a menos que el operando 1 y el operando 2 sean alguno de los siguientes: <ul style="list-style-type: none"> <li>• TIMESTAMP(aaaa)</li> <li>• TIMESTAMP(aaaaMM)</li> <li>• TIMESTAMP(MM)</li> </ul> En estos tres casos, el resultado es INTERVAL(aaaaMM)
DATE	-	TIMESTAMP	INTERVAL	INTERVAL(ddssmmffffff)
TIME STAMP	-	DATE	INTERVAL	INTERVAL(ddHHmmssffffff)
TIME STAMP	+/-	INTERVAL	TIMESTAMP	
INTERVAL	+	TIMESTAMP	TIMESTAMP	
DATE	+/-	INTERVAL	TIMESTAMP	
INTERVAL	+	DATE	TIMESTAMP	
INTERVAL	+/-	INTERVAL	INTERVAL	El operando 1 y el operando 2 deben tener ambos años y meses (como máximo) o días (como máximo) y un valor de hora
INTERVAL	*//	NUMBER	INTERVAL	

#### Consulta relacionada

- “Asignaciones” en la página 374
- “dateValue()” en la página 795
- “extend()” en la página 796

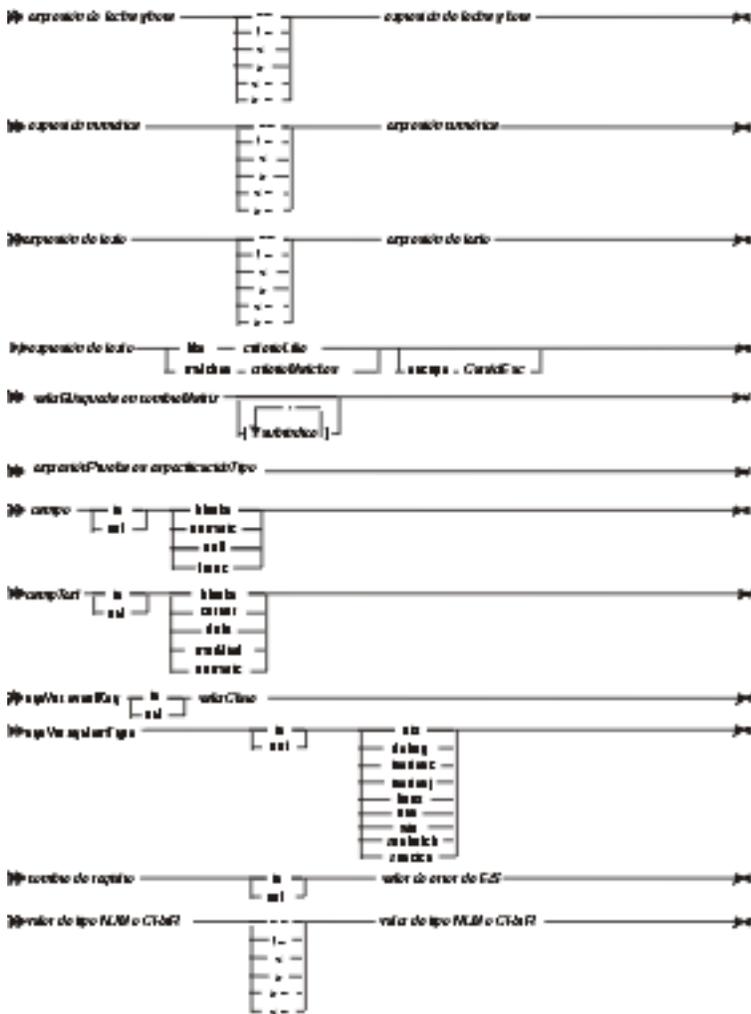
“intervalValue()” en la página 797  
“timeValue()” en la página 801  
“timeStampValue()” en la página 799  
“Expresiones” en la página 495  
“Expresiones lógicas”  
“Expresiones numéricas” en la página 504  
“Operadores y precedencia” en la página 673  
“Tipos primitivos” en la página 34  
“Expresiones de texto” en la página 505  
  
“Subseries” en la página 753

## Expresiones lógicas

Una *expresión lógica* se resuelve en true (verdadero) o false (falso) y se utiliza como criterio en una sentencia **if** o **while** o (en algunas situaciones) en una sentencia **case**.

### Expresiones lógicas elementales

Una expresión lógica elemental se compone de un operando, un operador de comparación y un segundo operador, como se muestra en el diagrama de sintaxis y en la tabla subsiguiente:



Primer operando	Operador de comparación	Segundo operando
<i>expresión de fecha y hora</i>	Uno de los siguientes: <code>==, !=, &lt;, &gt;, &lt;=, &gt;=</code>	<i>expresión de fecha y hora</i>  La primera y segunda expresiones deben ser de tipos compatibles.  En el caso de las comparaciones de fecha y hora, el signo mayor que (>) significa después en el tiempo y el signo menor que (<) significa antes en el tiempo.
<i>expresión numérica</i>	Uno de los siguientes: <code>==, !=, &lt;, &gt;, &lt;=, &gt;=</code>	<i>expresión numérica</i>
<i>expresión de serie</i>	Uno de los siguientes: <code>==, !=, &lt;, &gt;, &lt;=, &gt;=</code>	<i>expresión de serie</i>

Primer operando	Operador de comparación	Segundo operando
<i>expresión de serie</i>	like	<p><i>likeCriterion</i> que es un campo de caracteres o literal con el que se compara la <i>expresión de serie</i> por posición de carácter de izquierda a derecha. La utilización de esta característica es parecida a la utilización de la palabra clave <b>like</b> en consultas SQL.</p> <p><i>escChar</i> es un campo de un carácter o literal que se resuelve en un carácter de escape.</p> <p>Para conocer más detalles, consulte la sección <i>Operador like</i>.</p>
<i>expresión de serie</i>	matches	<p><i>matchCriterion</i> que es un campo de caracteres o literal con el que se compara la <i>expresión de serie</i> por posición de carácter de izquierda a derecha. La utilización de esta característica es parecida a la utilización de <i>expresiones regulares</i> en UNIX o Perl.</p> <p><i>escChar</i> es un campo de un carácter o literal que se resuelve en un carácter de escape.</p> <p>Para conocer más detalles, consulte la sección <i>Operador matches</i>.</p>
Valor de tipo NUM o CHAR, como el descrito para el segundo operando	Uno de los siguientes: ==, !=, <, >, <=, >=	<p>Valor de tipo NUM o CHAR, que puede ser uno de los siguientes:</p> <ul style="list-style-type: none"> <li>• Un campo de tipo NUM sin posiciones decimales</li> <li>• Un literal entero</li> <li>• Un campo o literal de tipo CHAR</li> </ul>
<i>searchValue</i>	in	<i>arrayName</i> ; para obtener detalles, consulte <i>in</i> .
<i>campo no de un registro SQL</i>	Uno de los siguientes: <ul style="list-style-type: none"> <li>• is</li> <li>• not</li> </ul>	Uno de los siguientes: <ul style="list-style-type: none"> <li>• blanks (para probar si el valor de un campo de caracteres es o no sólo blancos).</li> <li>• numeric (para probar si el valor de un campo de tipo CHAR o MBCHAR es o no numérico)</li> </ul>

Primer operando	Operador de comparación	Segundo operando
<i>campo de un registro SQL</i>	Uno de los siguientes: <ul style="list-style-type: none"> <li>• is</li> <li>• not</li> </ul>	Uno de los siguientes: <ul style="list-style-type: none"> <li>• blanks (para probar si el valor de un campo de caracteres es o no sólo blancos).</li> <li>• null (para probar si el campo se ha establecido en nulo mediante una sentencia set o leyendo desde una base de datos relacional)</li> <li>• numeric (para probar si el valor de un campo de tipo CHAR o MBCHAR es o no numérico)</li> <li>• trunc (para probar si se han suprimido caracteres no blancos a la derecha cuando un valor de carácter de un solo byte o de doble byte se ha leído por última vez desde una base de datos relacional en el campo)</li> </ul> <p>La prueba de truncamiento sólo puede resolverse en true si la columna de la base de datos es más larga que el campo. El valor de la prueba es false una vez que un valor se ha movido al campo o una vez que el campo se ha establecido en nulo.</p>
<i>campoTexto</i> (nombre de un campo de un formulario de texto)	Uno de los siguientes: <ul style="list-style-type: none"> <li>• is</li> <li>• not</li> </ul>	Uno de los siguientes: <ul style="list-style-type: none"> <li>• blanks (para probar si el valor del campo de texto está o no limitado a blancos u nulos).</li> </ul> <p>La prueba de blancos se basa en la última entrada realizada por el usuario en el formulario, no en el contenido actual del campo de formulario; una prueba que utilice <i>is</i> es true en estos casos:</p> <ul style="list-style-type: none"> <li>– La última entrada realizada por el usuario eran blancos o nulo; o</li> <li>– El usuario no ha especificado datos en el campo desde el inicio del programa o desde que se ejecutó una sentencia set de tipo <i>set form initial</i>.</li> </ul> <ul style="list-style-type: none"> <li>• cursor (para probar si el usuario ha dejado el cursor en el campo de texto especificado).</li> <li>• data (para probar si el campo de texto especificado contiene otros datos aparte de blancos o nulos).</li> <li>• modified (para probar si se ha establecido el código de datos modificados del campo, como se describe en el apartado Código y propiedad de datos modificados).</li> <li>• numeric (para probar si el valor de un campo de tipo CHAR o MBCHAR es o no numérico).</li> </ul>

Primer operando	Operador de comparación	Segundo operando
ConverseVar.eventKey	Uno de los siguientes: <ul style="list-style-type: none"> <li>• is</li> <li>• not</li> </ul>	Para obtener más detalles, consulte la sección <i>ConverseVar.eventKey</i> .
sysVar.systemType	Uno de los siguientes: <ul style="list-style-type: none"> <li>• is</li> <li>• not</li> </ul>	Para obtener detalles, consulte <i>sysVar.systemType</i> .  No puede utilizarse <i>is</i> ni <i>not</i> para probar un valor devuelto por <i>VGLib.getVAGSysType</i> .
<i>nombre de registro</i>	Uno de los siguientes: <ul style="list-style-type: none"> <li>• is</li> <li>• not</li> </ul>	Un valor de error de E/S adecuado a la organización del registro. Consulte el apartado <i>Valores de error de E/S</i> .

La tabla siguiente lista los operadores de comparación, cada uno de los cuales se utiliza en una expresión que se resuelve en true o false.

Operador	Finalidad
==	El operador <i>equality</i> indica si dos operandos tienen el mismo valor.
!=	El operador <i>not equal</i> indica si dos operandos tienen valores diferentes.
<	El operador <i>less than</i> indica si el primero de los dos operandos es numéricamente inferior al segundo.
>	El operador <i>greater than</i> indica si el primero de los dos operandos es numéricamente superior al segundo.
<=	El operador <i>less than or equal to</i> indica si el primero de los dos operandos es numéricamente inferior o igual al segundo.
>=	El operador <i>greater than or equal to</i> indica si el primero de los dos operandos es numéricamente superior o igual al segundo.
in	El operador <i>in</i> indica si el primero de los dos operandos es un valor del segundo, que hace referencia a una matriz. Para obtener detalles, consulte <i>in</i> .
is	El operador <i>is</i> indica si el primero de los dos operandos está en la categoría del segundo. Para obtener detalles, consulte la tabla anterior.
like	El operador <i>like</i> indica si los caracteres de los primeros dos operandos coincide con el segundo operando, tal como se describe en la sección <i>Operador like</i> .
matches	El operador <i>matches</i> indica si los caracteres de los primeros dos operandos coincide con el segundo operando, tal como se describe en la sección <i>Operador matches</i> .
not	El operador <i>not</i> indica si el primero de los dos operandos no está en la categoría del segundo. Para obtener detalles, consulte la tabla anterior.

La tabla siguiente y las descripciones que siguen indican las normas de compatibilidad cuando los operandos son de los tipos especificados.

Tipo primitivo del primer operando	Tipo primitivo del segundo operando
BIN	BIN, DECIMAL, FLOAT, MONEY, NUM, NUMC, PACF, SMALLFLOAT

Tipo primitivo del primer operando	Tipo primitivo del segundo operando
CHAR	CHAR, DATE, HEX, MBCHAR, NUM, TIME, TIMESTAMP
DATE	CHAR, DATE, NUM, TIMESTAMP
DBCHAR	DBCHAR
DECIMAL	BIN, DECIMAL, FLOAT, MONEY, NUM, NUMC, PACF, SMALLFLOAT
HEX	CHAR, HEX
MBCHAR	CHAR, MBCHAR
MONEY	BIN, DECIMAL, FLOAT, MONEY, NUM, NUMC, PACF, SMALLFLOAT
NUM	BIN, CHAR, DATE, DECIMAL, FLOAT, MONEY, NUM, NUMC, PACF, SMALLFLOAT, TIME
NUMC	BIN, DECIMAL, FLOAT, MONEY, NUM, NUMC, PACF, SMALLFLOAT
PACF	BIN, DECIMAL, FLOAT, MONEY, NUM, NUMC, PACF, SMALLFLOAT
TIME	CHAR, NUM, TIME, TIMESTAMP
TIMESTAMP	CHAR, DATE, TIME, TIMESTAMP
UNICODE	UNICODE

Los detalles son los siguientes:

- Un valor de cualquiera de los tipos numéricos (BIN, DECIMAL, FLOAT, MONEY, NUM, NUMC, PACF, SMALLFLOAT) puede compararse con un valor de cualquier tipo numérico y tamaño, y EGL realizará las conversiones temporales necesarias. Una comparación de igualdad de fracciones equivalentes (como 1.4 y 1.40) se evalúa en true, aunque las posiciones decimales sean diferentes.
- Un valor de tipo CHAR puede compararse con un valor de tipo HEX sólo si todos los caracteres de tipo CHAR están dentro del rango de dígitos hexadecimales (0-9, A-F, a-f). EGL convierte temporalmente las letras minúsculas a mayúsculas en el valor de tipo CHAR.
- Si una comparación incluye dos valores tipo carácter (CHAR, DBCHAR, HEX, MBCHAR, UNICODE) y un valor tiene menos bytes que el otro, una conversión temporal rellena el valor más corto por la derecha:
  - En una comparación con un valor de tipo MBCHAR, un valor de tipo CHAR se rellena por la derecha con blancos de un solo byte
  - En una comparación con un valor de tipo HEX, un valor de tipo CHAR se rellena por la derecha con ceros binarios
  - Un valor de tipo DBCHAR se rellena por la derecha con blancos de doble byte
  - Un valor de tipo UNICODE se rellena por la derecha con blancos Unicode de doble byte
  - Un valor de tipo HEX se rellena por la derecha con ceros binarios, lo que significa (por ejemplo) que si un valor "0A" debe expandirse a dos bytes, a efectos de comparación el valor será "0A00" en lugar de "000A"
- Un valor de tipo CHAR puede compararse con un valor de tipo NUM sólo si se cumplen estas condiciones:

- El valor de tipo CHAR contiene dígitos de un solo byte, sin otros caracteres
- La definición del valor de tipo NUM no tiene coma decimal

Una comparación entre CHAR y NUM funciona de la forma siguiente:

- Una conversión temporal coloca el valor NUM en un formato CHAR. Los caracteres numéricos se justifican por la izquierda, añadiendo blancos de un solo byte si es necesario. Si un campo de tipo NUM de longitud 4 tiene el valor 7, por ejemplo, el valor se tratará como "7" con tres blancos a la derecha.
- Si la longitud de los campos elementos no coincide, una conversión temporal rellena el valor más corto con blancos por la derecha.
- La comparación comprueba los valores byte por byte. Observe estos dos ejemplos:
  - Un campo de tipo CHAR de longitud 2 y valor "7 " (incluido un blanco) es igual a un campo de tipo NUM de longitud 1 y valor 7, ya que el campo temporal que se basa en el campo de tipo NUM también incluye un blanco final
  - Un campo de tipo CHAR de valor "8" es mayor que un campo de tipo NUM de valor 534, ya que el "8" va después del "5" en el orden de búsqueda de caracteres ASCII o EBCDIC

## Expresiones lógicas complejas

Puede crear una expresión más compleja utilizando un operador *and* (&&) u *or* (||) para combinar un par o varias expresiones elementales. Además, puede utilizar el operador *not* (!), como se describe más adelante.

Si una expresión lógica se compone de expresiones lógicas elementales combinadas mediante operadores *or*, EGL evalúa la expresión de acuerdo con las normas de precedencia, pero la detiene si una de las expresiones lógicas elementales se resuelve en true. Observe el ejemplo:

```
field01 == field02 || 3 in array03 || x == y
```

Si field01 no es igual a field02, la evaluación continúa. Sin embargo, si el valor 3 está en array03, la expresión global se evalúa en true, y la última expresión lógica elemental (x == y) no se evalúa.

De forma parecida, si se combinan expresiones lógicas elementales mediante operadores *and*, EGL detiene la evaluación si una de las expresiones lógicas elementales se resuelve en false. En el ejemplo siguiente, la evaluación se detiene en cuanto se averigua que field01 no es igual a field02:

```
field01 == field02 && 3 in array03 && x == y
```

Puede utilizar pares de paréntesis en una expresión lógica para cualquiera de los siguientes propósitos:

- Para cambiar el orden de evaluación.
- Para clarificar el significado.
- Para hacer posible la utilización del operador *not* (!), que se resuelve en un valor booleano (true o false) opuesto al valor de una expresión lógica que sigue inmediatamente. La expresión subsiguiente debe estar entre paréntesis.

## Ejemplos

Al consultar los ejemplos que siguen, suponga que value1 contiene "1", value2 contiene "2", y así sucesivamente:

```

/* == true */
value5 < value2 + value4

/* == false */
!(value1 es numérico)

/* == true cuando la salida generada se ejecuta
 en Windows 2000, Windows NT
 o z/OS UNIX System Services */
sysVar.systemType is WIN || sysVar.systemType is USS

/* == true */
(value6 < 5 || value2 + 3 >= value5) && value2 == 2

```

### Conceptos relacionados

“Código de datos modificados y propiedad modified” en la página 161

### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### Consulta relacionada

“case” en la página 566  
 “Expresiones de fecha y hora” en la página 495  
 “Manejo de excepciones” en la página 94  
 “Expresiones” en la página 495  
 “Valores de error de E/S” en la página 536  
 “if, else” en la página 608  
 “operador in” en la página 532  
 “Operador like” en la página 656  
 “Operador like” en la página 656  
 “Expresiones numéricas”  
 “Operadores y precedencia” en la página 673  
 “Tipos primitivos” en la página 34  
 “Expresiones de texto” en la página 505  
 “eventKey” en la página 923  
 “getVAGSysType()” en la página 919  
 “systemType” en la página 937  
 “while” en la página 648

## Expresiones numéricas

Una *expresión numérica* se resuelve en un número. Una expresión de este tipo puede especificarse en varias situaciones; por ejemplo, en el lado derecho de una sentencia assignment. Una expresión numérica puede estar formada por:

- Un operando numérico, que es uno de los siguientes:
  - Una variable que contiene un número. El elemento puede ir precedido de un signo.
  - Un literal numérico, que puede empezar con un signo, pero siempre tiene una serie de dígitos y puede incluir una única coma decimal.
  - Una invocación de función que devuelve un número.

El tipo de un literal numérico está implícito en el valor del literal:

- Un entero de 4 dígitos o menos es de tipo SMALLINT
- Un entero de 5 a 8 dígitos es de tipo INT
- Un entero de 9 a 18 dígitos es de tipo BIGINT
- Un número que incluye una coma decimal es de tipo NUM

- Un operando numérico, seguido de un operador numérico, seguido de un segundo operando numérico.
- Una expresión más compleja que se forma utilizando un operador numérico para combinar un par de expresiones más básicas.

Puede utilizar pares de paréntesis en una expresión numérica para cambiar el orden de evaluación o aclarar el significado.

Al revisar los ejemplos siguientes, suponga que `intValue1` es igual a 1, `intValue2` es igual a 2, etc., y que cada valor no tiene posiciones decimales:

```
/* == -8, con los paréntesis alterando temporalmente
 la preferencia habitual de * y + */
intValue2 * (intValue1 - 5)
```

```
/* == -2, con un menos único como último operador */
intValue2 + -4
```

```
/* == 1.4, si la expresión se asigna a un
 elemento con al menos una posición decimal. */
intValue7 / intValue5
```

```
/* == 2, que es un resto
 expresado como un valor entero */
intValue7 % intValue5
```

En la sección *Expresiones* encontrará un ejemplo que muestra el efecto de los paréntesis sobre la utilización de un signo más (+).

Una expresión numérica puede dar un resultado inesperado si un valor calculado intermedio requiere más de 128 bits.

#### Consulta relacionada

“Expresiones de fecha y hora” en la página 495

“Expresiones” en la página 495

“Expresiones lógicas” en la página 497

“Operadores y precedencia” en la página 673

“Tipos primitivos” en la página 34

“Expresiones de texto”

## Expresiones de texto

Una *expresión de texto* se resuelve en una serie de caracteres y puede especificarse en diversas situaciones; por ejemplo, en el lado derecho de una sentencia de asignación (assignment). La expresión de texto puede ser cualquiera de las siguientes:

- Una variable que contiene una serie de caracteres.
- Un *literal de serie*, que es una serie de caracteres delimitada por comillas. El literal es de tipo `STRING`.
- Una subserie de un literal o variable que contenga una serie de caracteres. Para obtener detalles, consulte *Subseries*.
- La invocación de cualquier palabra del sistema de formateo de series que devuelva una serie de caracteres. Para obtener detalles, consulte *Formateo de series (palabras del sistema)*.
- Una serie de valores de las clases anteriores, donde cada valor está separado del siguiente por el operador de concatenación que es un signo más (+). La sentencia siguiente asigna *WebSphere* a `myString`:

```
myString = "Web" + "Sphere";
```

En la sección *Expresiones* encontrará un ejemplo que muestra el efecto de los paréntesis sobre la utilización de un signo más (+).

- Cualquier otra invocación de función que devuelva una serie de caracteres.

Cualquier carácter precedido del carácter de escape (\) se incluirá en la expresión de texto. En particular, puede utilizar el carácter de escape para incluir los siguientes caracteres en un literal, campo o valor de retorno:

- Signo de comillas (")
- Barra inclinada invertida (\)
- Retroceso, indicado por \b
- Salto de hoja, indicado por \f
- Carácter de línea nueva, indicado mediante \n
- Retorno de carro, indicado mediante \r
- Tabulador, indicado por \t

Ejemplos:

```
myString = "Dijo, \iEscapa mientras puedas!\i";
myString2 = "¿Es necesaria una barra inclinada invertida (\\)?";
```

Si un literal no tiene comillas de cierre, se produce un error:

```
myString3 = "Es imposible escapar\i";
```

Cada valor de la expresión de texto debe ser válido para el contexto en el que se utiliza la expresión. Por ejemplo, un elemento de tipo UNICODE no puede utilizarse en una expresión asignada a un elemento de tipo CHAR. Encontrará más detalles en *Asignaciones*.

#### Consulta relacionada

- “Asignaciones” en la página 374
- “Expresiones de fecha y hora” en la página 495
- “Expresiones” en la página 495
- “Expresiones lógicas” en la página 497
- “Expresiones numéricas” en la página 504
- “Operadores y precedencia” en la página 673
- “Tipos primitivos” en la página 34
  
- “Subseries” en la página 753

---

## Formato del archivo plugin.xml del descriptor de construcción maestro

El archivo plugin.xml del descriptor de construcción maestro es un archivo XML que el entorno de trabajo utiliza para especificar el nombre y el nombre de vía de acceso a archivo del descriptor de construcción maestro. Solamente lo necesita si necesita un descriptor de construcción maestro para imponer ciertas opciones a utilizar para la generación y está generando desde el entorno de trabajo o está utilizando el mandato EGLCMD. Debe colocar este archivo plugin.xml en un directorio en el directorio plugins. El formato del archivo es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
 id="id"
 name="plg"
 version="5.0"
 vendor-name="com">
```

```

<requires />
<runtime />
<extension point =
"com.ibm.etools.egl.generation.base.framework.masterBuildDescriptor">
 <masterBuildDescriptor file = "bfil" name = "mas" />
</extension>
</plugin>

```

donde:

*id* El identificador del conector

*plg*  
El nombre del conector

*com*  
El nombre de su empresa

*bfil*  
El nombre de vía de acceso de un archivo que contenga un descriptor de construcción maestro, con el formato *proyecto/carpeta/archivo*, relativo al directorio de espacio de trabajo de Enterprise Developer, donde:

*proyecto*  
El nombre del directorio del proyecto

*carpeta*  
El nombre de un directorio dentro del directorio del proyecto

*archivo*  
El nombre de un archivo que contiene un descriptor de construcción maestro

*mas*  
El nombre de un descriptor de construcción maestro

El contenido de este archivo debe seguir las normas de un archivo XML. Para separar nombres de archivo dentro de un nombre de vía de acceso debe utilizar el carácter de barra inclinada (/).

Debe especificar el atributo de nombre y el atributo de archivo. De lo contrario, se ignorará el archivo plugin.xml.

A continuación se muestra un ejemplo del contenido del archivo plugin.xml:

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Ejemplo de conector BuildDescriptor maestro -->

<plugin
 id="example.master.BuildDescriptor.plugin"
 name="Example master BuildDescriptor plug-in"
 version="5.0"
 vendor-name="IBM">
 <requires />
 <runtime />
 <!-- ===== -->
 <!-- -->
 <!-- Registre el BuildDescriptor maestro -->
 <!-- -->
 <!-- ===== -->
 <extension point =
"com.ibm.etools.egl.generation.base.framework.masterBuildDescriptor" >
 <masterBuildDescriptor file
= "myProject/myFolder/myFile.eglbld" name = "masterBD" />
 </extension>
</plugin>

```

### Conceptos relacionados

“Componente descriptor de construcción” en la página 293

“Descriptor de construcción maestro” en la página 297

### Tareas relacionadas

“Generar desde la interfaz por lotes del entorno de trabajo” en la página 331

“Generar en el entorno de trabajo” en la página 329

### Consulta relacionada

“Opciones del descriptor de construcción” en la página 382

“EGLCMD” en la página 478

“Formato del archivo eglmaster.properties” en la página 490

---

## Componente FormGroup en formato fuente EGL

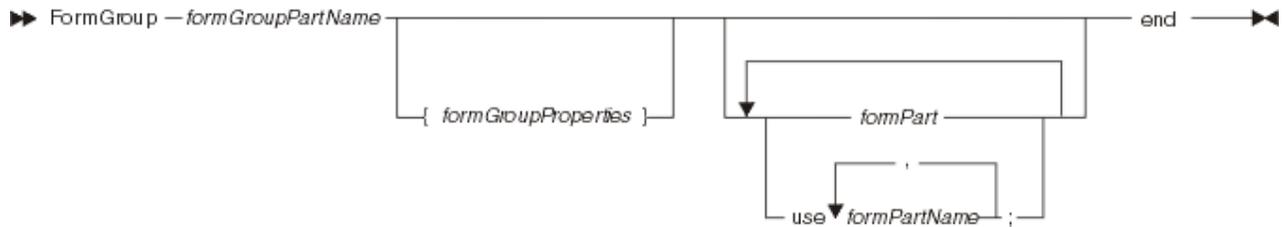
Un componente formGroup se declara en un archivo EGL, que está descrito en *Formato fuente EGL*. Este componente es un componente primario, lo que significa que debe estar en el nivel superior del archivo y debe tener el mismo nombre que el archivo.

Un programa solamente puede utilizar formularios que estén asociados con un grupo de formularios a los que hace referencia la declaración use del programa.

Este es un ejemplo de un componente formGroup:

```
FormGroup myFormGroup
{
 validationBypassKeys = [pf3],
 helpKey = "pf1",
 pfKeyEquate = yes,
 screenFloatingArea
 {
 screenSize = [24,80],
 topMargin = 0,
 bottomMargin = 0,
 leftMargin = 0,
 rightMargin = 0
 },
 printFloatingArea
 {
 pageSize = [60,80],
 topMargin = 3,
 bottomMargin = 3,
 leftMargin = 5,
 rightMargin = 5
 }
}
use myForm01;
use myForm02;
end
```

El diagrama de un componente formGroup es el siguiente:



**FormGroup** *nombreComponenteGrupoFormularios ... end*

Identifica el componente como un grupo de formularios y especifica el nombre de componente. Para conocer las reglas de denominación, consulte *Convenios de denominación*.

*propiedadesGrupoFormularios*

Una serie de propiedades, cada una separada de la siguiente por una coma. Cada propiedad se describe más adelante.

*componenteFormulario*

Un formulario de texto o impresión, tal como se describe en *Componente de formulario en formato fuente EGL*.

**use** *nombreComponenteFormulario*

Una declaración use que proporciona acceso a un formulario que no está incorporado en el grupo de formularios.

Las propiedades del grupo de formularios son las siguientes:

**alias**

Una serie incorporada a los nombres de la salida generada. Si no especifica un alias, se utilizará en su lugar el nombre del componente `formGroup` .

**validationBypassKeys** = [*valorTeclaSalto*]

Identifica una o varias pulsaciones de usuario que provocan que el entorno de ejecución de EGL se salte las validaciones de campos de entrada. Esta propiedad es de utilidad para reservar una pulsación que finalice el programa rápidamente. Cada opción de *valorTeclaSalto* es como se indica a continuación:

**pf*n***

El nombre de una tecla F o PF, incluido un número entre 1 y 24.

**Nota:** Las teclas de función de un teclado de PC son con frecuencia teclas *F*, como por ejemplo F1, pero EGL utiliza la terminología IBM *PF* a fin de que (por ejemplo) F1 se denomine PF1.

Si desea especificar más de un valor de tecla, delimite el conjunto de valores con corchetes y separe cada uno de los valores del siguiente con una coma, como en el ejemplo siguiente:

```
validationBypassKeys = [pf3, pf4]
```

**helpKey** = "*valorTeclaAyuda*"

Identifica una pulsación de usuario que provoca que el entorno de ejecución de EGL presente al usuario un formulario de ayuda. La opción *valorTeclaAyuda* es como se indica a continuación:

**pf*n***

El nombre de una tecla f o pf, incluido un número entre 1 y 24.

**Nota:** Las teclas de función de un teclado de PC suelen ser teclas *f* tales como *f1*, pero EGL utiliza la terminología de IBM *pf* de forma que, por ejemplo, *f1* se denomina *pf1*.

**pfKeyEquate = yes, pfKeyEquate = no**

Especifica si la pulsación que se registra cuando el usuario pulsa una tecla de función con un número alto (de PF13 a PF24) es la misma que la pulsación registrada cuando el usuario pulsa una tecla de función inferior a 12. Encontrará los detalles en *pfKeyEquate*.

**screenFloatingArea { propiedades }**

Define el área flotante utilizada para la salida a una pantalla. Para obtener una visión general de las áreas flotantes, consulte *Componente de formulario*. Para conocer detalles sobre las propiedades, consulte la siguiente sección.

**printFloatingArea { propiedades }**

Define el área flotante utilizada para la salida imprimible. Para obtener una visión general de las áreas flotantes, consulte *Componente de formulario*. Para conocer los detalles de propiedades, consulte *Propiedades de un área flotante de impresión*.

## Propiedades de un área flotante de pantalla

El conjunto de propiedades tras **screenFloatingArea** está delimitado por corchetes ( `{ }` ) y cada propiedad está separada de la siguiente por una coma. Las propiedades son las siguientes:

**screenSize = [filas, columnas]**

El número de filas y columnas en el área de presentación en línea, incluidas las líneas o columnas utilizadas como márgenes. El valor por omisión es el siguiente:

```
screenSize=[24,80]
```

**topMargin= filas**

El número de líneas en blanco en la parte superior del área de presentación. El valor por omisión es 0.

**bottomMargin= filas**

El número de líneas en blanco en la parte inferior del área de presentación. El valor por omisión es 0.

**leftMargin= columnas**

El número de columnas en blanco a la izquierda del área de presentación. El valor por omisión es 0.

**rightMargin= columnas**

El número de columnas en blanco a la derecha del área de presentación. El valor por omisión es 0.

## Propiedades de un área flotante de impresión

El conjunto de propiedades tras **printFloatingArea** está delimitado por corchetes ( `{ }` ) y cada propiedad está separada de la siguiente por una coma. Las propiedades son las siguientes:

**pageSize = [filas, columnas]**

El número de filas y columnas en el área de presentación imprimible, incluidas las líneas o columnas utilizadas como márgenes. Esta propiedad es necesaria si especifica un área flotante de impresión.

**deviceType = singleByte, deviceType = doubleByte**

Especifica si la declaración de área flotante es para una impresora que da

soporte a salida de un solo byte (el valor por omisión) o a salida de doble byte. Especifique **doubleByte** si alguno de los formularios incluye elementos del tipo DBCHAR o MBCHAR.

**topMargin = *filas***

El número de líneas en blanco en la parte superior del área de presentación. El valor por omisión es 0.

**bottomMargin = *filas***

El número de líneas en blanco en la parte inferior del área de presentación. El valor por omisión es 0.

**leftMargin = *columnas***

El número de columnas en blanco a la izquierda del área de presentación. El valor por omisión es 0.

**rightMargin = *columnas***

El número de columnas en blanco a la derecha del área de presentación. El valor por omisión es 0.

**Conceptos relacionados**

“Proyectos, paquetes y archivos EGL” en la página 13

“Componente FormGroup” en la página 153

“Componente de formulario” en la página 154

**Consulta relacionada**

“Formato fuente EGL” en la página 491

“Componente de formulario en formato fuente EGL”

“Convenios de denominación” en la página 672

“pfKeyEquate” en la página 686

“Declaración use” en la página 954

---

## Componente de formulario en formato fuente EGL

Un componente de formulario se declara en un archivo EGL, como se describe en el apartado *Formato fuente EGL*. Si sólo un grupo de formularios accede a un componente de formulario, es aconsejable que éste esté incorporado al componente formGroup. Si varios grupos grupo de formularios acceden a un componente de formulario, es necesario especificar el componente de formulario en el nivel superior de un archivo EGL.

A continuación se ofrece un ejemplo de formulario de texto:

```
Form myTextForm type textForm
{
 formsize= [24, 80],
 position= [1, 1],
 validationBypassKeys=[pf3, pf4],
 helpKey="pf1",
 helpForm="myHelpForm",
 msgField="myMsg",
 alias = "form1"
}

* { position=[1, 31], value="Sample Menu" } ;
* { position=[3, 18], value="Activity:" } ;
* { position=[3, 61], value="Command Code:" } ;

activity char(42)[5] { position=[4,18], protect=skip } ;

commandCode char(10)[5] { position=[4,61], protect=skip } ;
```

```

* { position=[10, 1], value="Response:" } ;
response char(228) { position=[10, 12], protect=skip } ;

* { position=[13, 1], value="Command:" } ;
myCommand char(70) { position=[13,10] } ;

* { position=[14, 1], value="Enter=Run F3=Exit" } ;

myMsg char(70) { position=[20,4] } ;

end

```

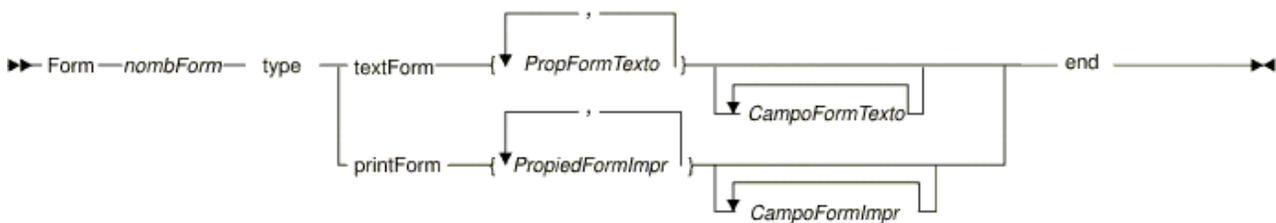
A continuación se ofrece un ejemplo de formulario de impresión:

```

Form myPrintForm type printForm
{
 formsize= [48, 80],
 position= [1, 1],
 msgField="myMsg",
 alias = "form2"
}
* { position=[1, 10], value="Your ID: " } ;
ID char(70) { position=[1, 30] } ;
myMsg char(70) { position=[20, 4] } ;
end

```

El diagrama de un componente de formulario es el siguiente:



### **Form** nombreFormulario ... end

Identifica el componente como formulario y especifica el nombre del componente. Para conocer las normas de denominación, consulte el apartado Convenios de denominación.

### **textForm**

Indica que el formulario es de texto.

### *propiedadFormularioTexto*

Una propiedad de formulario de texto. Para obtener detalles, consulte el apartado *Formulario de texto*.

### *campoFormularioTexto*

Un campo de formulario de texto. Para obtener detalles, consulte el apartado *Campos de formulario*.

### **printForm**

Indica que el formulario es de impresión.

### *propiedadFormularioImpresión*

Una propiedad de formulario de impresión. Para obtener detalles, consulte el apartado *Formulario de impresión*.

### *campoFormularioImpresión*

Un campo de formulario de impresión. Para obtener detalles, consulte el apartado *Campos de formulario*.

## Propiedades de formulario de texto

Las propiedades de formulario de texto son las siguientes:

**formSize** = [*filas, columnas*]

Número de filas y columnas del área de presentación en línea. Esta propiedad es obligatoria.

El valor de columna es equivalente al número de caracteres de un solo byte que pueden visualizarse en el área de presentación.

**position** = [*fila, columna*]

Fila y columna en las que se visualiza el formulario en el área de presentación. Si omite esta propiedad, el formulario será flotante y se visualizará en el área flotante, en la próxima línea libre en la que quepa todo el formulario del área flotante.

**validationBypassKeys** = [*valorTeclaSalto*]

Identifica una o varias pulsaciones de teclas que provocan que el entorno de ejecución EGL pase por alto validaciones de campos de entrada. Esta propiedad resulta de utilidad para reservar una pulsación que finaliza el programa con rapidez. La opción *valorTeclaSalto* es la siguiente:

**pf*n***

El nombre de una tecla F o PF, que incluye un número entre 1 y 24

**Nota:** Las teclas de función de un teclado de PC son con frecuencia teclas *F*, como por ejemplo F1, pero EGL utiliza la terminología IBM *PF* a fin de que (por ejemplo) F1 se denomine PF1.

Si desea especificar más de un valor de tecla, delimite el conjunto de valores con paréntesis y separe cada uno de los valores del siguiente con una coma, como en el ejemplo siguiente:

```
validationBypassKeys = [pf3, pf4]
```

**helpKey** = "*valorTeclaAyuda*"

Identifica una pulsación de tecla del usuario que provoca que el entorno de ejecución EGL presente un formulario de ayuda al usuario. La opción *valorTeclaAyuda* es la siguiente:

**pf*n***

El nombre de una tecla F o PF, que incluye un número entre 1 y 24

**Nota:** Las teclas de función de un teclado de PC son con frecuencia teclas *F*, como por ejemplo F1, pero EGL utiliza la terminología IBM *PF* a fin de que (por ejemplo) F1 se denomine PF1.

**helpForm** = "*nombreFormulario*"

Nombre del formulario de ayuda específico del formulario de texto.

**msgField** = "*nombreCampo*"

Nombre del campo de formulario de texto que visualiza un mensaje en respuesta a un error de validación o en respuesta a la ejecución de `ConverseLib.displayMsgNum`.

**alias** = "*alias*"

Un alias de 8 caracteres como máximo, destinado al entorno de ejecución EGL.

## Propiedades de formulario de impresión

Las propiedades de formulario de impresión son las siguientes:

**formsize** = [filas, columnas]

Número de filas y columnas del área de presentación en línea. Esta propiedad es obligatoria.

El valor de columna es equivalente al número de caracteres de un solo byte que pueden visualizarse en el área de presentación.

**position** = [fila. columna]

Fila y columna en las que se visualiza el formulario en el área de presentación. Si omite esta propiedad, el formulario será flotante y se visualizará en el área flotante, en la próxima línea libre en la que quepa todo el formulario del área flotante.

**msgField** = "nombreCampo"

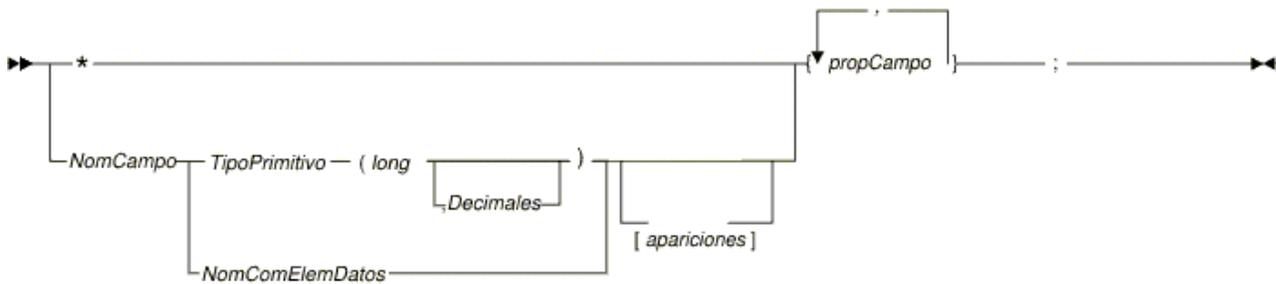
Nombre del campo de formulario de texto que visualiza un mensaje en respuesta a la ejecución de `ConverseLib.displayMsgNum`.

**alias** = "alias"

Un alias de 8 caracteres como máximo, destinado al entorno de ejecución EGL.

## Campos de formulario

El diagrama de un campo de formulario es el siguiente:



\* Indica que el campo es de constante. No tiene ningún nombre sino un valor de constante, que se especifica en la propiedad **value** específica del campo. Las sentencias del código no pueden acceder al valor de un campo de constante.

*propiedadCampo*

Una propiedad del campo de formulario de texto. Para obtener información detallada, consulte el apartado *Propiedades del campo de formulario de texto*.

*nombreCampo*

Especifica el nombre del campo. Para conocer las normas de denominación, consulte el apartado *Convenios de denominación*.

El código puede acceder al valor de un campo con nombre, también llamado *campo de variable*.

Si un formulario de texto contiene un campo de variable que empieza en una línea y finaliza en otra, el formulario de texto sólo podrá visualizarse en pantallas cuya anchura sea igual a la del formulario.

*apariciones*

El número de elementos de una matriz de campos. Sólo están soportadas las matrices unidimensionales. Para obtener detalles, consulte el apartado *Para matrices de campos*.

*tipoPrimitivo*

El tipo primitivo asignado al campo. Esta especificación afecta a la longitud máxima, pero cualquier campo numérico se generará como de tipo NUM.

Los formularios que contienen campos de tipo DBCHAR sólo pueden utilizarse en sistemas y dispositivos que den soporte a juegos de caracteres de doble byte. De forma similar, los formularios que contienen campos de tipo MBCHAR sólo pueden utilizarse en sistemas y dispositivos que den soporte a juegos de caracteres multibyte.

Los tipos primitivos FLOAT, SMALLFLOAT y UNICODE no están soportados para formularios de texto ni de impresión.

#### *longitud*

La longitud del campo, que es un entero que representa el número máximo de caracteres o dígitos que puede contener el campo.

#### *decimales*

Para un tipo numérico (BIN, DECIMAL, NUM, NUMC o PACF), puede especificar *decimales*, que es un entero que representa el número de posiciones después de la coma decimal. El número máximo de posiciones decimales es el menor de dos números: 18 o el número de dígitos declarado como *longitud*. La coma decimal no se almacena con los datos.

#### *nombreComponenteElementoDatos*

El nombre de un componente dataItem que actúa como modelo de formato del campo, como se describe en *typeDef*. El componente dataItem debe ser visible para el componente de formulario, como se describe en el apartado *Referencias a componentes*.

## Propiedades del campo de formulario de texto

Las propiedades que sólo son útiles en los campos de formulario de texto se describen más adelante. Las siguientes propiedades se utilizan más frecuentemente y también están disponibles:

- “align” en la página 690
- “currency” en la página 694
- “currencySymbol” en la página 694
- “dateFormat” en la página 695
- “fillCharacter” en la página 699
- “isBoolean” en la página 701
- “lineWrap” en la página 704
- “lowerCase” en la página 705
- “masked” en la página 705
- “numericSeparator” en la página 709
- “outline” en la página 709
- “sign” en la página 712
- “timeFormat” en la página 715
- “timeStampFormat” en la página 716
- “upperCase” en la página 717
- “zeroFormat” en la página 722

### Para cualquier campo

Las siguientes propiedades son útiles para cualquier campo de un formulario:

**position** = [*fila. columna*]

Fila y columna del byte de atributo que precede al campo. Esta propiedad es obligatoria.

**value = "literalSerie"**

Una serie de caracteres que se visualiza en el campo. Las comillas son necesarias.

Esta propiedad puede especificarse para cualquier elemento; por ejemplo, en una declaración de componente `dataItem`.

**Nota:** Si está en vigor la compatibilidad de VisualAge Generator y establece la propiedad de formulario de texto **value**, el contenido de esa propiedad estará disponible en el programa sólo después de que el usuario haya devuelto el formulario. Por esta razón, no es necesario que el valor establecido en el programa sea válido para el elemento del programa.

**fieldLen = longitudEnBytes**

Longitud del campo; el número de caracteres de un solo byte que pueden visualizarse en el campo. Este valor no incluye el byte de atributo que lo precede.

El valor de **fieldLen** para campos numéricos debe ser lo suficientemente grande para visualizar el número mayor que el campo pueda contener, además de una coma decimal (si el número tiene posiciones decimales). El valor de **fieldLen** para un campo de tipo CHAR, DBCHAR, MBCHAR o UNICODE debe ser lo suficientemente grande para contener los caracteres de doble byte, así como los caracteres de desplazamiento a teclado ideográfico y a teclado estándar.

El valor por omisión de **fieldLen** es el número de bytes necesario para visualizar el mayor número posible para el tipo primitivo, incluidos todos los caracteres de formato.

## Para campos de texto variables

Las siguientes propiedades son útiles para campos de texto variables:

**cursor = no, cursor = yes**

Indica si el cursor de la pantalla se encuentra al principio del campo cuando el formulario se visualiza por primera vez. Sólo un campo del formulario puede tener la propiedad `cursor` establecida en *yes*. El valor por omisión es *no*.

**modified = no, modified = yes**

Indica si el programa considerará el campo como modificado, independientemente de que el usuario haya cambiado el valor. Para obtener detalles, consulte el apartado *Código de datos modificados y propiedad modified*.

El valor por omisión es *no*.

**protect = no, protect = skip, protect = yes**

Especifica si el usuario puede acceder al campo. Los valores válidos son los siguientes:

**no (el valor por omisión para los campos de variable)**

Establece el campo de forma que el usuario pueda sobrescribir el valor en él.

**skip (el valor por omisión para los campos de constante)**

Establece el campo de forma que el usuario no pueda sobrescribir el valor en él. Además, el cursor pasa por alto el campo en cualquiera de estos casos:

- El usuario trabaja en el campo anterior por orden de tabulación y pulsa el **tabulador** o rellena ese campo anterior con contenido; o
- El usuario trabaja en el campo siguiente por orden de tabulación y pulsa **Mayúsculas Tabulador**.

**yes**

Establece el campo de forma que el usuario no pueda escribir encima del valor que haya en él.

**validationOrder = entero**

Indica la posición del campo en el orden de validación. El orden por omisión en el que se validan los campos es el orden de los campos en la pantalla, de izquierda a derecha y de arriba a abajo.

## Para matrices de campos

Las matrices unidimensionales están soportadas en los formularios de texto y de impresión. En una declaración de matriz, el valor de la propiedad **occurs** es mayor que 1, como en este ejemplo:

```
myArray char(1)[3];
```

Los elementos de la matriz se colocan en relación a la situación especificada para el primer elemento de la matriz. El comportamiento por omisión es colocar los elementos verticalmente en filas consecutivas.

Utilice las siguientes propiedades para variar el comportamiento por omisión:

**columns = númeroDeElementos**

Número de elementos de matriz en cada fila. El valor por omisión es 1.

**linesBetweenRows = númeroDeLíneas**

Número de líneas entre cada fila que contiene elementos de matriz. El valor por omisión es 0.

**spacesBetweenColumns = númeroDeEspacios**

Número de espacios entre cada elemento de matriz. El valor por omisión es 1.

**indexOrientation = down, indexOrientation = across**

Especifica cómo hace referencia el programa a los elementos de una matriz.

- Si establece **indexOrientation** en *down*, los elementos se numeran de arriba a abajo y luego de izquierda a derecha, de forma que los elementos de una columna determinada queden numerados secuencialmente. Por omisión, el valor de **indexOrientation** es *down*.
- Si establece **indexOrientation** en *across*, los elementos se numeran de izquierda a derecha y luego de arriba a abajo, de forma que los elementos de una fila determinada queden numerados secuencialmente.

Puede alterar temporalmente ciertas propiedades de un elemento de matriz. En la siguiente declaración de campo, por ejemplo, la propiedad **cursor** se altera temporalmente en el segundo elemento de myArray:

```
myArray char(10)[5]
{position=[4,61], protect=skip, myArray[2] { cursor = yes } };
```

## Conceptos relacionados

“Código de datos modificados y propiedad modified” en la página 161

“Visión general de las propiedades de EGL” en la página 64

“Formularios de impresión” en la página 156

“Referencias a componentes” en la página 21

“Formularios de texto” en la página 158

“Typedef” en la página 27

## Consulta relacionada

“Propiedades de presentación de campos” en la página 66

“Propiedades de formato” en la página 66

“Convenios de denominación” en la página 672

“NUM” en la página 51

“Tipos primitivos” en la página 34

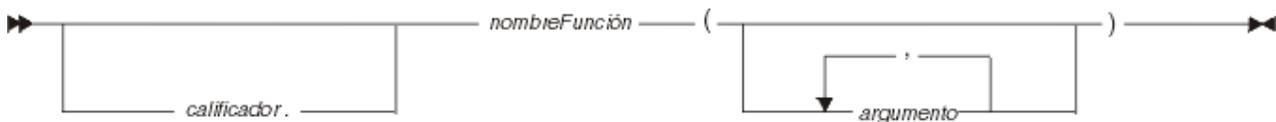
“displayMsgNum()” en la página 789

“Propiedades de validación” en la página 67

---

## Invocaciones de función

Una invocación de función ejecuta una función generada por EGL o una función de sistema. Cuando la función invocada finaliza, el proceso continúa con la sentencia que sigue a la invocación o (en casos complejos) con el siguiente proceso necesario en una expresión o en una lista de argumentos.



### *calificador*

Uno de los símbolos siguientes:

- El nombre de la biblioteca en la que reside la función; o
- El nombre del paquete en el que reside la función, seguido opcionalmente de un punto y del nombre de la biblioteca en la que reside la función.
- *this* (identifica una función del programa actual)

Para obtener detalles acerca de las circunstancias en las que el calificador es innecesario, consulte el apartado *Referencias a componentes*.

### *nombre de función*

Nombre de la función invocada.

### *argumento*

Uno de los siguientes:

- Literal
- Constante
- Variable
- Una expresión numérica, de texto o de fecha y hora más compleja, que puede incluir una invocación de función o una subserie; sin embargo, el modificador de acceso para el parámetro debe ser IN

El efecto de una variable que se pasa como un argumento a una función generada por EGL depende de si el parámetro correspondiente se modifica con IN, OUT o INOUT. Para obtener detalles, consulte la sección *Parámetros de función*.

Si la función invocada devuelve un valor, puede utilizar la invocación de estas formas:

- Como una sentencia EGL completa (en cuyo caso la función no devuelve un valor y va seguida de un punto y coma.)
- Como el valor origen en una sentencia de asignación.
- Como un operando en una expresión.
- Como un argumento en la invocación de una función

Una función invocada como en una invocación de función puede provocar un efecto secundario consistente en que una variable cambie de valor cuando se utilice la misma variable en la función o incluso en la misma invocación de

función. Observe este ejemplo, en el que se supone que la función Sum devuelve la suma de tres argumentos y que la función Increment añade uno a un argumento pasado:

```
b INT = 1;
x INT = Sum(Increment(b), b, Increment(b));
```

Si el argumento de Increment está relacionado con un parámetro modificado con INOUT, el efecto de las sentencias precedentes es el siguiente:

- b = 1
- La primera invocación (más a la izquierda) de Increment revisa el valor de b, que es 2 en el retorno de Increment
- El segundo argumento de la invocación de Sum es 2
- La segunda (más a la derecha) invocación de Increment revisa el valor de b, que es 3 en el retorno de Increment
- Después de la ejecución de Sum, x recibe el valor 7 porque la lógica de esa función ha utilizado los valores 2, 2 y 3

Si el segundo argumento de la invocación de Sum está relacionado con un parámetro modificado con INOUT, la evaluación de ese argumento se produce después de ambas invocaciones de Increment. El efecto del código precedente es el siguiente:

- b = 1
- La primera invocación (más a la izquierda) de Increment revisa el valor de b, que es 2 en el retorno de Increment
- La segunda (más a la derecha) invocación de Increment revisa el valor de b, que es 3 en el retorno de Increment
- La lógica de Sum empieza a ejecutarse y solo entonces se asocia la memoria al segundo argumento referido; el valor en esa memoria es igual a 3
- Después de la ejecución de Sum, x recibe el valor 8 porque la lógica de esa función ha utilizado los valores 2, 3 y 3

La regla general es que los efectos secundarios pueden identificarse por referencia al orden habitual de evaluación de expresiones que es de izquierda a derecha pero que puede alterarse temporalmente mediante paréntesis. La utilización de INOUT es una complicación más, tal como se muestra.

Si el modificador de acceso de un parámetro es IN o OUT, las normas de compatibilidad se describen en el apartado *Compatibilidad de asignación*. Si el modificador de acceso de un parámetro es INOUT (o si el parámetro se encuentra en la función onPageLoad de un pageHandler), las normas de compatibilidad se describen en el apartado *Compatibilidad de referencia*.

También se aplican otras normas:

#### **literales**

Si el modificador de acceso es IN o INOUT, puede codificar un literal como argumento. El código generado por EGL crea una variable temporal del tipo de parámetro, inicializa esa variable con el valor y la pasa a la función.

#### **registro fijo**

Si el argumento es un registro fijo, el parámetro debe ser un registro fijo.

Los registros fijos que no son de tipo basicRecord están sujetos a las siguientes normas:

- El tipo del argumento y del parámetro deben ser idénticos
- El modificador de acceso debe ser de tipo INOUT

En relación a los registros fijos de tipo `basicRecord`, el tipo del argumento y del parámetro pueden variar:

- Si el modificador de acceso es de tipo `N`, la longitud del argumento debe ser mayor o igual que la longitud del parámetro.
- Si el modificador de acceso es de tipo `OUT` o `INOUT`, la longitud del argumento debe ser menor o igual que la longitud del parámetro.

### Conceptos relacionados

“Componente de función” en la página 140

“Referencias a componentes” en la página 21

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### Tareas relacionadas

“Asignaciones” en la página 374

### Consulta relacionada

“Compatibilidad de asignación en EGL” en la página 369

“Sentencias EGL” en la página 88

“Parámetros de función” en la página 522

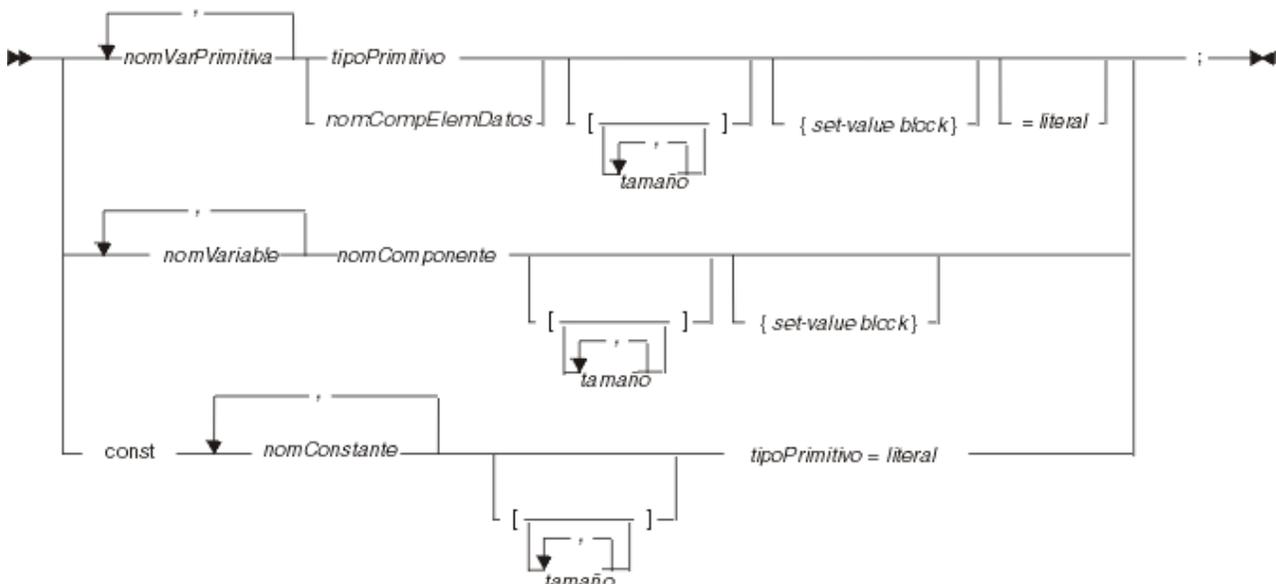
“Componente de función en formato fuente EGL” en la página 527

“Tipos primitivos” en la página 34

“Compatibilidad de referencia en EGL” en la página 739

## Variables de función

El diagrama de sintaxis para cada variable en una función es el siguiente:



#### *nombreVarPrim*

Especifica el nombre de una variable primitiva local. Para obtener detalles sobre la utilización de la función, consulte el apartado *Referencias a variables y constantes*. Para conocer otras reglas, consulte *Convenios de denominación*.

#### *tipoPrimitivo*

Tipo de un campo primitivo. En función del tipo, puede ser necesaria la siguiente información:

- La longitud del parámetro, que es un entero que representa el número de caracteres o dígitos del área de memoria.
- Para algunos tipos numéricos puede especificar un entero que represente el número de posiciones después de la coma decimal. La coma decimal no se almacena con los datos.
- Para un elemento de tipo INTERVAL o TIMESTAMP, puede especificar una máscara de fecha y hora, que asigna un significado (como por ejemplo "dígito de año") a una posición dada en el valor de elemento.

#### *nombreComponenteDataItem*

El nombre de un componente dataItem que es visible al programa. Para obtener detalles acerca de la visibilidad, consulte el apartado *Referencias a componentes*.

El componente actúa como un modelo de formato, tal como se describe en *Typedef*.

#### *tamaño*

El número de elementos de la matriz. Si especifica el número de elementos, la matriz se inicializa con ese número de elementos.

#### *bloque set-value*

Para obtener información detallada, consulte los apartados *Visión general de las propiedades de EGL* y *Bloques set-value*.

#### *= literal*

Especifica el valor inicial la variable primitiva.

#### *nombreVar*

Nombre de la variable, que puede ser de cualquier tipo basado en un componente.

#### *nombreComponente*

Nombre de un componente que es visible al programa o está predefinido. Para obtener detalles acerca de la visibilidad, consulte el apartado *Referencias a componentes*.

El componente actúa como un modelo de formato, tal como se describe en *Typedef*.

#### **const** *nombreConstante tipoPrimitivo=literal*

Nombre, tipo y valor de una constante. Especifique una serie entrecomillada (para un tipo de carácter); un número (para un tipo numérico); o una matriz de valores del tipo adecuado (para una matriz). Ejemplos:

```
const myString String = "Great software!";
const myArray BIN[] = [36, 49, 64];
const myArray02 BIN[][] = [[1,2,3],[5,6,7]];
```

Para conocer las reglas de denominación, consulte *Convenios de denominación*.

#### **Conceptos relacionados**

"Componente de función" en la página 140

"Componentes" en la página 17

"Referencias a componentes" en la página 21

"Referencias a variables en EGL" en la página 58

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

"Typedef" en la página 27

#### **Tareas relacionadas**

"Componente de función en formato fuente EGL" en la página 527

### Consulta relacionada

“Matrices” en la página 74

“INTERVAL” en la página 42

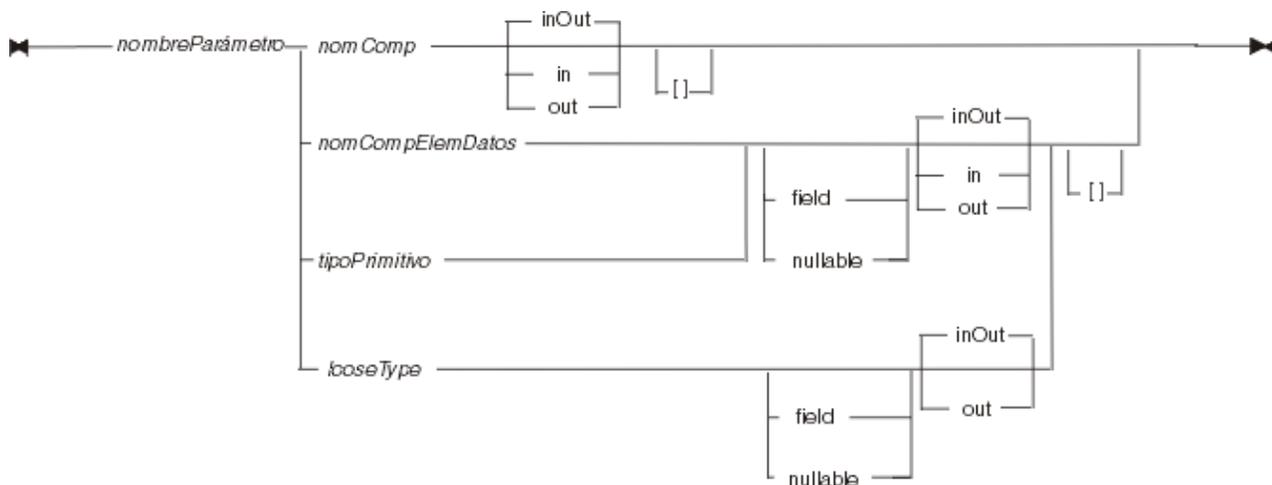
“Convenios de denominación” en la página 672

“TIMESTAMP” en la página 44

---

## Parámetros de función

El diagrama de sintaxis para un parámetro de función es el siguiente:



### nombreParámetro

Especifica el nombre de un parámetro, que puede ser un registro o un elemento de datos; o una matriz de registros o elementos de datos. Para conocer las reglas, consulte *Convenios de denominación*.

Si especifica el modificador **inOut** o **out**, los cambios realizados en el valor del parámetro están disponibles en la función invocadora. Los modificadores se describen posteriormente y en la sección “Implicaciones de inOut y los modificadores relacionados” en la página 525.

Un parámetro no es visible en las funciones que invoca la función que contiene el parámetro; sin embargo, un parámetro se puede pasar como argumento de aquellas otras funciones.

Un parámetro que termina en corchetes ( [ ] ) es una matriz dinámica, y las demás especificaciones declaran aspectos de cada elemento de esa matriz.

### inOut

La función recibe el valor del argumento como una entrada y el invocador recibe los cambios en el parámetro cuando termina la función. Sin embargo, si el argumento es un literal o una constante, el argumento se trata como si el modificador **in** estuviera en vigor.

El modificador **inOut** es necesario si el parámetro es un elemento y si especifica el modificador **field**, lo que indica que el parámetro tiene atributos de campo de formulario con posibilidad de prueba como *blanks* o *numeric*.

Si el parámetro es un registro (no un registro fijo), el modificador **inOut** es el único válido.

Si el parámetro es un registro fijo, se aplican las normas siguientes:

- Si intenta utilizar ese registro para acceder a un archivo o una base de datos en la función actual (o en una función invocada por la función actual), debe especificar el modificador **inOut** o aceptar ese modificador por omisión
- Si el tipo de registro es el mismo para el argumento y el parámetro (por ejemplo si ambos son registros de serie) la información de estado específica de registro como por ejemplo el estado de final de archivo está disponible en la función y se devuelve al invocador, pero sólo si el modificador **inOut** está en vigor

Si el modificador **inOut** está en vigor, el argumento relacionado debe tener compatibilidad de referencia con el parámetro, como se describe en el apartado *Compatibilidad de referencia en EGL*.

**in** La función recibe el valor del argumento como una entrada, pero el invocador no se ve afectado por cambios realizados en el parámetro.

No puede utilizar el modificador **in** para un elemento que tenga el modificador **field**. Además, no puede especificar el modificador **in** para un registro (que no sea un registro fijo); o para un registro fijo que se utilice para acceder a un archivo o una base de datos en la función actual o en una función invocada por la función actual.

**out**

La función no recibe el valor del argumento como una entrada; en lugar de esto, el valor de entrada se inicializa según las reglas descritas en la sección *Inicialización de datos*. El valor del parámetro se asigna al argumento cuando la función efectúa el retorno.

Si el argumento es un literal o una constante, el argumento se trata como si el modificador **in** estuviera en vigor.

No puede utilizar el modificador **out** para un parámetro que tenga el modificador **field**. Tampoco puede especificar el modificador **out** para un registro, o para un registro fijo que se utilice para acceder a un archivo o una base de datos en la función actual o en una función invocada por la función actual.

*nombreComponente*

Un componente de registro que es visible para la función y que actúa como typedef (un modelo de formato) de un parámetro. Para obtener detalles acerca de los componentes que son visibles, consulte el apartado *Referencias a componentes*.

La entrada y salida (E/S) de un registro fijo está sujeta a las siguientes consideraciones:

- Un registro fijo pasado desde otra función en el mismo programa incluye información de estado de registro, como por ejemplo el valor de error de E/S *endOfFile*, pero sólo si el registro es del mismo tipo que el parámetro. De forma parecida, cualquier cambio en el estado del registro se devuelve al llamador, de modo que, si realiza una operación de E/S en un parámetro del registro, las pruebas realizadas en ese registro pueden producirse en la función actual, en el llamador o en una función a la que llame la función actual.

Las funciones de biblioteca no reciben información de estado de registro.

- Cualquier operación de E/S realizada en el registro fijo utiliza las propiedades de registro especificadas para el parámetro, no las propiedades de registro especificadas para el argumento.

- Para los registros fijos de tipo `indexedRecord`, `mqRecord`, `relativeRecord` o `serialRecord`, el archivo o cola de mensajes asociada con la declaración de registro se trata como recurso de la unidad de ejecución en lugar de como recurso de programa. Las declaraciones de registro locales comparten el mismo archivo (o cola) siempre que la propiedad de registro `fileName` (o `queueName`) tiene el mismo valor. Sólo puede asociarse un archivo físico simultáneamente con un nombre de archivo o cola, independientemente de cuántos registros estén asociados con el archivo o cola en la unidad de ejecución, y EGL refuerza esta norma cerrando y reabriendo los archivos según convenga.

#### *nombreComponenteElementoDatos*

Un componente `dataItem` que es visible para la función y que actúa como `typedef` (un modelo de formato) de un parámetro.

#### *tipoPrimitivo*

Tipo de un campo primitivo. En función del tipo, puede ser necesaria la siguiente información:

- La longitud del parámetro, que es un entero que representa el número de caracteres o dígitos del área de memoria.
- Para algunos tipos numéricos puede especificar un entero que represente el número de posiciones después de la coma decimal. La coma decimal no se almacena con los datos.
- Para un elemento de tipo `INTERVAL` o `TIMESTAMP`, puede especificar una máscara de fecha y hora, que asigna un significado (como por ejemplo "dígito de año") a una posición dada en el valor de elemento.

#### *tipoLoose*

Un tipo `loose` es una clase especial de tipo primitivo que sólo se utiliza para parámetros de función. Este tipo se utiliza si se desea que el parámetro acepte un rango de longitudes de argumento. La ventaja consiste en que puede invocar la función repetidamente y pasar un argumento de longitud diferente cada vez.

Los valores válidos son los siguientes:

- `CHAR`
- `DBCHAR`
- `HEX`
- `MBCHAR`
- `NUMBER`
- `UNICODE`

Si desea que el parámetro acepte un número de cualquier tipo primitivo y longitud, especifique `NUMBER` como tipo `loose`. En este caso, el número pasado al parámetro no debe tener posiciones decimales.

Si desea que el parámetro acepte una serie de un tipo primitivo determinado pero cualquier longitud, especifique `CHAR`, `DBCHAR`, `MBCHAR`, `HEX` o `UNICODE` como tipo `loose` y asegúrese de que el argumento es del tipo primitivo correspondiente.

La definición del argumento determina lo que ocurre cuando una sentencia de la función opera sobre un parámetro de tipo `loose`.

Los tipos `loose` no están disponibles en funciones declaradas en *bibliotecas*.

Para obtener detalles acerca de los tipos primitivos, consulte el apartado *Tipos primitivos*.

### **field**

Indica que el parámetro tiene atributos de campo de formulario, como por ejemplo *blanks* o *numeric*. Dichos atributos pueden probarse en una expresión lógica.

El modificador **field** solo está disponible si especifica el modificador **inOut** o acepta el modificador **inOut** predeterminado.

El modificador **field** no está disponible para parámetros de función en una biblioteca de tipo `nativeLibrary`.

### **nullable**

Indica las siguientes características del parámetro:

- El parámetro puede establecerse en nulo
- El parámetro tiene acceso a la información de estado necesaria para probar el truncamiento o el establecimiento en nulo en una expresión lógica

El modificador **nullable** sólo es significativo si el argumento pasado al parámetro es un elemento de estructura de un registro SQL. Se aplican las siguientes normas:

- El parámetro puede establecerse en nulo y comprobarse para nulos sólo si la propiedad de elemento **isNullable** está establecida en *yes*
- La capacidad para probar el truncamiento está disponible independientemente del valor de **isNullable**
- Puede especificar **nullable** independientemente de si el modificador **inOut**, **in** o **out** está en vigor.

## **Implicaciones de inOut y los modificadores relacionados**

Para conocer mejor los modificadores **inOut**, **out** e **in**, revise el ejemplo siguiente, que muestra (en comentarios) los valores de distintas variables en distintos puntos de ejecución.

```
program inoutpgm
a int;
b int;
c int;

function main()
a = 1;
b = 1;
c = 1;

func1(a,b,c);

// a = 1
// b = 3
// c = 3
end

function func1(x int in, y int out, z int inout)
// a = 1 x = 1
// b = 1 y = 0
// c = 1 z = 1

x = 2;
y = 2;
z = 2;

// a = 1 x = 2
```

```

// b = 1 y = 2
// c = 2 z = 2

func2();
func3(x, y, z);
// a = 1 x = 2
// b = 1 y = 3
// c = 3 z = 3

end

function func2()
// a = 1
// b = 1
// c = 2

end

function func3(q int in, r int out, s int inout)
// a = 1 x = unresolved q = 2
// b = 1 y = unresolved r = 2
// c = 2 z = unresolved s = 2

q = 3;
r = 3;
s = 3;

// a = 1 x = unresolved q = 3
// b = 1 y = unresolved r = 3
// c = 3 z = unresolved s = 3

end

```

### Conceptos relacionados

"Componente de función" en la página 140  
 "Componente de biblioteca de tipo basicLibrary" en la página 142  
 "Componente de biblioteca de tipo basicLibrary" en la página 142  
 "Componentes" en la página 17  
 "Referencias a componentes" en la página 21  
 "Referencias a variables en EGL" en la página 58  
 "Typedef" en la página 27

### Consulta relacionada

"Componente de registro básico en formato fuente EGL" en la página 379  
 "Inicialización de datos" en la página 471  
 "Formato fuente EGL" en la página 491  
 "Componente de función en formato fuente EGL" en la página 527  
 "Componente de registro indexado en formato fuente EGL" en la página 535  
 "INTERVAL" en la página 42  
 "Expresiones lógicas" en la página 497  
 "Componente de registro MQ en formato fuente EGL" en la página 662  
 "Convenios de denominación" en la página 672  
 "Tipos primitivos" en la página 34  
 "Compatibilidad de referencia en EGL" en la página 739  
 "Componente de registro relativo en formato fuente EGL" en la página 740  
 "Componente de registro serie en formato fuente EGL" en la página 743  
 "Componente de registro SQL en formato fuente EGL" en la página 748  
 "TIMESTAMP" en la página 44

---

## Componente de función en formato fuente EGL

Puede declarar funciones en un archivo fuente EGL, como se describe en el apartado *Formato fuente EGL*.

El ejemplo siguiente muestra un componente de programa con dos funciones incorporadas, junto con una función autónoma y un componente de registro autónomo:

```
Program myProgram(employeeNum INT)
{includeReferencedFunctions = yes}

// variable global de programa
employees record_ws;
employeeName char(20);

// función incorporada obligatoria
Function main()

// inicializar nombres de empleados
recd_init();

// obtener el nombre de empleado correcto
// en función del employeeNum pasado
employeeName = getEmployeeName(employeeNum);
end

// otra función incorporada
Function recd_init()
employees.name[1] = "Employee 1";
employees.name[2] = "Employee 2";
end
end

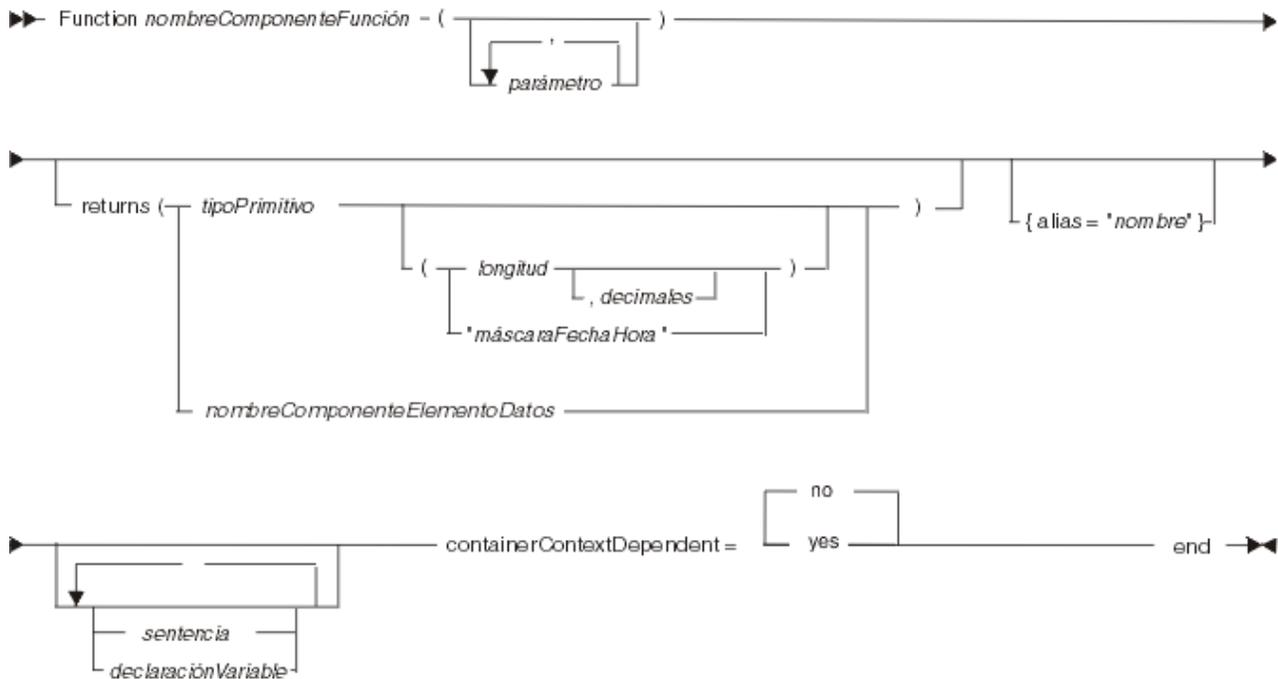
// función autónoma
Function getEmployeeName(employeeNum INT) returns (CHAR(20))

// variable local
index BIN(4);
index = syslib.size(employees.name);
if (employeeNum > index)
return("Error");
else
return(employees.name[employeeNum]);
end

end

// componente de registro que actúa como typeDef de empleados
Record record_ws type basicRecord
10 name CHAR(20)[2];
end
```

El diagrama de sintaxis de un componente de función es el siguiente:



### Function *nombreComponenteFunción* ... end

Identifica el componente como función y especifica el nombre del componente. Para conocer las normas de denominación, consulte el apartado *Convenios de denominación*.

#### *parámetro*

Un parámetro es un área de memoria disponible en toda la función y que puede recibir un valor de la función invocadora. Para obtener detalles sobre la sintaxis utilizada para declarar un parámetro, consulte *Parámetros de función*.

#### returns ( *tipoRetorno* )

Describe los datos devueltos por la función al invocante. Las características del tipo de retorno deben coincidir con las características de la variable que recibe el valor en la función invocante.

#### { **alias** = *nombre* }

Solo es válido si la función está en una biblioteca de tipo `nativeLibrary`. En ese contexto, *nombre* es el nombre de la función basada en DLL y por omisión es el nombre de la función EGL. Establezca la propiedad **alias** explícitamente si se produce un error de validación cuando denomine la función EGL con el nombre de la función basada en DLL.

#### *nombreComponenteElementoDatos*

Un componente `dataItem` que es visible para la función y que actúa como `typedef` (un modelo de formato) del valor de retorno.

#### *tipoPrimitivo*

El tipo primitivo de los datos devueltos al invocante.

#### *longitud*

La longitud de los datos devueltos al invocante. La longitud es un entero que representa el número de caracteres o dígitos del valor devuelto.

#### *decimales*

Para algunos tipos numéricos puede especificar *decimales* que es un entero que representa el número de posiciones después de la coma decimal. El número

máximo de posiciones decimales es el menor de dos números: 18 o el número de dígitos declarado como *longitud*. La coma decimal no se almacena con los datos.

*" máscaraFechaHora"*

Para los tipos `TIMESTAMP` e `INTERVAL` puede especificar *máscaraFechaHora*, que asigna un significado (como por ejemplo "dígito de año") a una posición determinada en el valor de fecha y hora. La máscara no se almacena con los datos.

*sentencia*

Una sentencia EGL, según se describe en el apartado *Sentencias EGL*. La mayoría terminan con un signo de punto y coma.

*declaraciónVariable*

Una declaración de variable, tal como se describe en la sección *Variables de función*.

*containerContextDependent*

Una indicación de si se debe ampliar el espacio de nombres utilizado para resolver las funciones invocadas por la función que se declara. El valor por omisión es *no*.

Este indicador es para utilizarlo en el código migrado de VisualAge Generator. Para obtener detalles, consulte el apartado *containerContextDependent*.

**Conceptos relacionados**

"Proyectos, paquetes y archivos EGL" en la página 13

"Componente de función" en la página 140

"Import" en la página 33

"Componente de biblioteca de tipo `basicLibrary`" en la página 142

"Componente de biblioteca de tipo `basicLibrary`" en la página 142

"Componentes" en la página 17

"Referencias a componentes" en la página 21

"Referencias a variables en EGL" en la página 58

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

"Typedef" en la página 27

**Consulta relacionada**

"Matrices" en la página 74

"`containerContextDependent`" en la página 465

"Sentencias EGL" en la página 88

"Invocaciones de función" en la página 518

"Parámetros de función" en la página 522

"Variables de función" en la página 520

"`INTERVAL`" en la página 42

"Valores de error de E/S" en la página 536

"Convenios de denominación" en la página 672

"Tipos primitivos" en la página 34

"`TIMESTAMP`" en la página 44

---

## Salida generada

La tabla siguiente lista la salida generada. Para obtener información detallada sobre los nombres asignados a cada tipo de archivo de salida, consulte *Salida generada (referencia)*.

Tipo de salida	Finalidad	Tipo de generación
Plan de construcción	Lista los pasos de preparación de código que se realizarán en la plataforma destino	Java o envoltura Java
Bean de sesión EJB (Enterprise JavaBean)	Se ejecuta en un contenedor EJB	Envoltura Java
Programa Java y clases relacionadas	Se ejecuta fuera de J2EE o contexto de una aplicación cliente J2EE, aplicación Web o contenedor EJB	Java
Envoltura Java	Invoca un programa generado por EGL desde el código Java no generado por EGL	Envoltura Java
Archivo de entorno J2EE	Proporciona entradas para insertarlas en el descriptor de despliegue Java	Java
Biblioteca (salida generada)	Proporciona funciones y valores para que los utilice otra salida generada	Java
Archivo de propiedades de enlace	Controla cómo se realizan las llamadas a partir del código Java generado, pero sólo si las decisiones son finales durante el despliegue y no durante la generación	Java o envoltura Java
Componente PageHandler	Crea salida que controla la interacción de ejecución de un usuario con una página Web	Java
Archivo de propiedades del programa	Contiene propiedades de entorno de ejecución Java en un formato que sólo es accesible cuando se depura un programa Java en un proyecto Java no J2EE	Java
Archivo de resultados	Proporciona información de estado sobre los pasos de preparación de código que se han realizado en la plataforma destino	Java o envoltura Java

#### Conceptos relacionados

“Introducción a EGL” en la página 1

“Programa Java, PageHandler y biblioteca” en la página 328

“Propiedades de tiempo de ejecución Java” en la página 347

“Configuraciones de tiempo de ejecución” en la página 9

#### Tareas relacionadas

“Construir la salida de EGL” en la página 327

#### Consulta relacionada

“Salida generada (referencia)”

---

## Salida generada (referencia)

La salida de la generación EGL depende en gran medida de si se genera Java o una envoltura Java. La tabla siguiente muestra los nombres de archivo de la salida generada que no proviene de un componente EGL específico.

Tipo de salida	Nombre de archivo
“Plan de construcción” en la página 327	<i>aliasBuildPlan.xml</i>

“Bean de sesión EJB (Enterprise JavaBean)” en la página 315	<i>alias</i> EJBHome.java para la interfaz inicial, <i>alias</i> EJB.java para la interfaz de beans remota y <i>alias</i> EJBBean.java para la implementación de beans
“Archivo de entorno J2EE” en la página 357	<i>alias</i> -env.txt
“Archivo de propiedades del programa” en la página 350	<i>alias</i> .properties
“Archivo de resultados” en la página 328	<i>alias</i> _Results_ <i>indicaciónHora</i> .xml

### *alias*

El *alias*, si existe, que se ha especificado en el componente de programa. Si no se especifica el *alias*, se utiliza el nombre del componente de programa pero se trunca (si es necesario) al número máximo de caracteres permitidos en el entorno de ejecución.

Otras características de *alias* están determinadas por la clase de salida:

- Si está generando un programa Java, cada letra de *alias* tomará sin cambios las mismas mayúsculas o minúsculas que el código fuente
- Si está generando una envoltura de Java, las normas para denominar la envoltura y el bean de sesión EJB son las siguientes:
  - La primera letra del *alias* es mayúscula
  - Cada letra siguiente es minúscula, con esta excepción: se eliminan los subrayados o guiones y la letra siguiente es mayúscula

### *indicaciónHora*

La fecha y hora de creación del archivo. El formato refleja los valores en el sistema operativo de desarrollo.

Para obtener detalles sobre los nombres de archivo, consulte el tema de referencia adecuado:

- 
- “Salida de la generación de programa Java” en la página 675
- “Salida de la generación de envoltura Java” en la página 676

### Conceptos relacionados

“Plan de construcción” en la página 327

“Bean de sesión EJB (Enterprise JavaBean)” en la página 315

“Salida generada” en la página 529

“Generación” en la página 323

“Archivo de entorno J2EE” en la página 357

“Archivo de propiedades del programa” en la página 350

“Archivo de resultados” en la página 328

### Consulta relacionada

“Salida de la generación de programa Java” en la página 675

“Salida de la generación de envoltura Java” en la página 676

---

## Vista Resultados de la generación

La vista Resultados de la generación le muestra mensajes de preparación de código que son el resultado de la generación realizada en el entorno de trabajo. Estos mensajes pueden ser errores, avisos o mensajes informativos. Esta vista solamente está disponible al generar desde el entorno de trabajo. El formato es el siguiente:

*msgid mensaje*

**msgid**

Es el identificador del mensaje. Por ejemplo, IWN.VAL.4610.e es el ID de mensaje para el número de error de validación 4610 de Enterprise Developer.

**mensaje**

Es el texto del mensaje.

Los resultados de la generación se visualizan en la vista según el componente primario (programa, PageHandler, grupo de formularios, tabla de datos, biblioteca), con una pestaña diferente para cada uno. Los resultados pueden ser una combinación de resultados de validación y resultados de generación.

Puede abrir esta vista en cualquier momento, pero visualiza datos solamente después de que haya generado salida.

**Conceptos relacionados**

“Proceso de desarrollo” en la página 8

“Salida generada” en la página 529

“Generación” en la página 323

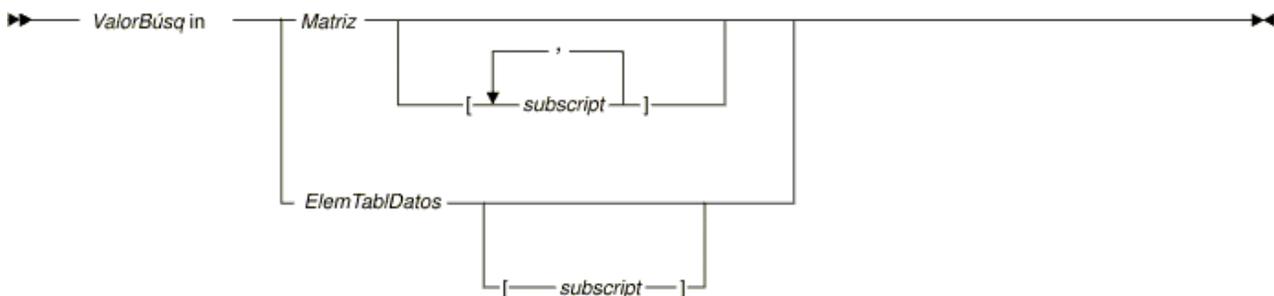
**Consulta relacionada**

“Salida generada (referencia)” en la página 530

---

## operador in

El operador **in** es un operador binario utilizado en una expresión lógica elemental que tiene el siguiente formato:



*valorBúsqueda*

Un literal o elemento, pero no una variable de sistema.

**matriz** Una matriz unidimensional o multidimensional. El operador **in** opera en una matriz unidimensional, que puede ser un elemento de una matriz multidimensional.

**subíndice**

Un entero o un elemento (o variable de sistema) que se resuelve en un entero. El valor de un subíndice es un índice que hace referencia a un elemento específico de una matriz.

Un elemento utilizado como subíndice de una matriz no puede ser por sí mismo un elemento de matriz. En cada uno de los ejemplos siguientes, `myItemB[1]` es tanto un subíndice como un elemento de matriz; como resultado, la siguiente sintaxis *no* es válida:

```
/* la sintaxis siguiente no es válida */
myItemA[myItemB[1]]

// esta sintaxis no es válida; pero sólo
// debido a que myItemB es myItemB[1], el
// primer elemento de una matriz unidimensional
myItemA[myItemB]
```

### **dataTableItem**

El nombre de un elemento `dataTable`. El elemento representa una columna de la tabla de datos. El operador **in** interactúa con esa columna como si ésta fuera una matriz unidimensional.

La expresión lógica se resuelve en `true` si el programa generado encuentra el valor de búsqueda. La búsqueda empieza en el elemento identificado por el último subíndice de matriz. Si *matriz* es una matriz unidimensional, el último subíndice es opcional y toma por omisión el valor 1. Si *matriz* es una matriz multidimensional, se aplican las siguientes normas:

- Debe estar presente un subíndice para cada dimensión
- El programa generado busca en la matriz unidimensional identificada por la secuencia de subíndices que no sean el último subíndice
- La búsqueda empieza en el elemento identificado por el último subíndice

En relación a las matrices tanto unidimensionales como multidimensionales, la búsqueda finaliza en el último elemento de la matriz unidimensional bajo revisión.

La expresión lógica que incluye **in** se resuelve en `false` en los siguientes casos:

- El valor de búsqueda no se encuentra
- El valor del último subíndice es mayor que el número de entradas de la matriz unidimensional en la que se busca

Si la expresión lógica elemental se resuelve en `true`, la operación **in** establece la variable de sistema `sysVar.arrayIndex` en el valor de subíndice del elemento que contiene el valor de búsqueda. Si la expresión se resuelve en `false`, la operación establece `sysVar.arrayIndex` en cero.

## **Ejemplos con una matriz unidimensional**

Supongamos que el elemento de estructura `myString` está subestructurado en una matriz de tres caracteres:

```
structureItem name="myString" length=3
structureItem name="myArray" occurs=3 length=1
```

La tabla siguiente muestra el efecto del operador **in** si `myString` es "ABC".

Expresión lógica	Valor de la expresión	Valor de sysVar.arrayIndex	Comentario
"A" en myArray	true	1	El subíndice de una matriz unidimensional toma por omisión el valor 1

Expresión lógica	Valor de la expresión	Valor de sysVar. ArrayIndex	Comentario
"C" en myArray[2]	true	3	La búsqueda empieza en el segundo elemento
"A" en myArray[2]	false	0	La búsqueda finaliza en el último elemento

## Ejemplos con una matriz multidimensional

Supongamos que la matriz myArray01D está subestructurada en una matriz de tres caracteres:

```
structureItem name="myArray01D" occurs=3 length=3
structureItem name="myArray02D" occurs=3 length=1
```

En este ejemplo, myArray01D es una matriz unidimensional, cada uno de cuyos elementos contiene una serie que está subestructurada en una matriz de tres caracteres. myArray02D es una matriz bidimensional, cada uno de cuyos elementos (como por ejemplo myArray02D[1,1]) contiene un solo carácter.

Si el contenido de myArray01D es "ABC", "DEF" y "GHI", el contenido de myArray02D es el siguiente:

```
"A" "B" "C"
"D" "E" "F"
"G" "H" "I"
```

La tabla siguiente muestra el efecto del operador **in**.

Expresión lógica	Valor de la expresión	Valor de sysVar. ArrayIndex	Comentario
"DEF" en myArray01D	true	2	Una referencia a una matriz unidimensional no requiere un subíndice; por omisión, la búsqueda empieza en el primer elemento
"C" en myArray02D[1]	—	—	La expresión no es válida debido a que una referencia a una matriz multidimensional debe incluir un subíndice para cada dimensión
"I" en myArray02D[3,2]	true	3	La búsqueda empieza en la tercera fila, segundo elemento
"G" en myArray02D[3,2]	false	0	La búsqueda finaliza en el último elemento de la fila revisada
"G" en myArray02D[2,4]	false	0	El segundo subíndice es mayor que el número de columnas disponibles para la búsqueda

### Tareas relacionadas

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

### Consulta relacionada

"Matrices" en la página 74

"Expresiones lógicas" en la página 497

"Operadores y precedencia" en la página 673

"arrayIndex" en la página 929

---

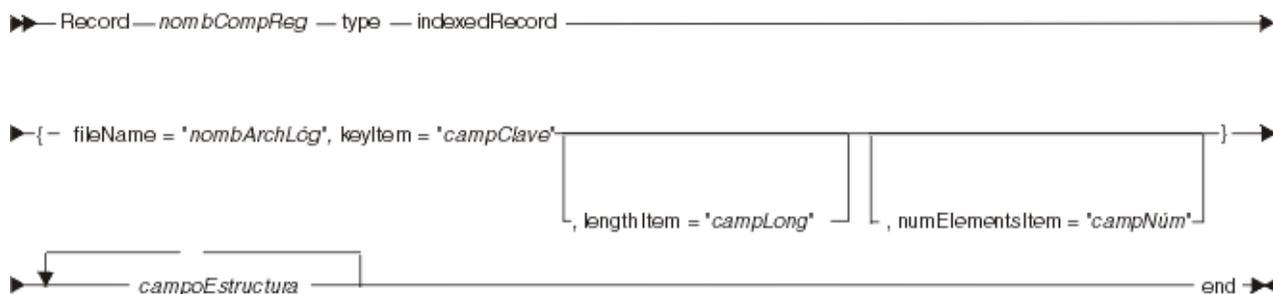
## Componente de registro indexado en formato fuente EGL

Un componente de registro de tipo `indexedRecord` se declara en un archivo EGL, como se describe en el apartado *Formato fuente EGL*.

A continuación se ofrece un ejemplo de componente de registro indexado:

```
Record myIndexedRecordPart type indexedRecord
{
 fileName = "myFile",
 keyItem = "myKeyItem"
}
10 myKeyItem CHAR(2);
10 myContent CHAR(78);
end
```

El diagrama de sintaxis de un componente de registro indexado es el siguiente:



### Record nombreComponenteRegistro indexedRecord

Identifica el componente como de tipo `indexedRecord` y especifica el nombre. Para conocer las normas, consulte el apartado *Convenios de denominación*.

### fileName = "nombreArchivoLógico"

El nombre de archivo. Para obtener detalles acerca del significado de la entrada, consulte el apartado *Asociaciones de recursos (visión general)*. Para conocer las normas, consulte el apartado *Convenios de denominación*.

### keyItem = "elementoClave"

El elemento de clave, que sólo puede ser un elemento de estructura exclusivo en el mismo registro. Debe utilizar una referencia no calificada para `keyItem`; por ejemplo, utilice `myItem` en lugar de `myRecord.myItem`. (En una función, sin embargo, puede hacer referencia a ese elemento de estructura al igual que haría con cualquier elemento de estructura).

### lengthItem = "elementoLongitud"

El elemento de longitud, como se describe en el apartado *Propiedades que dan soporte a registros de longitud variable*.

**numElementsItem** = "elementoNúmeroElementos"

El elemento de número de elementos, como se describe en el apartado *Propiedades que dan soporte a registros de longitud variable*.

*elementoEstructura*

Un elemento de estructura, como se describe en el apartado *Elemento de estructura en formato fuente EGL*.

### Conceptos relacionados

"Proyectos, paquetes y archivos EGL" en la página 13

"Referencias a componentes" en la página 21

"Componentes" en la página 17

"Componentes de registro" en la página 132

"Referencias a variables en EGL" en la página 58

"Asociaciones de recursos y tipos de archivo" en la página 304

"Typedef" en la página 27

### Tareas relacionadas

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

### Consulta relacionada

"Matrices" en la página 74

"Componente DataItem en formato fuente EGL" en la página 472

"Formato fuente EGL" en la página 491

"Componente de función en formato fuente EGL" en la página 527

"Componente de registro MQ en formato fuente EGL" en la página 662

"Convenios de denominación" en la página 672

"Tipos primitivos" en la página 34

"Componente de programa en formato fuente EGL" en la página 728

"Propiedades que dan soporte a registros de longitud variable" en la página 737

"Componente de registro relativo en formato fuente EGL" en la página 740

"Componente de registro serie en formato fuente EGL" en la página 743

"Componente de registro SQL en formato fuente EGL" en la página 748

"Elemento de estructura en el formato fuente de EGL" en la página 752

---

## Valores de error de E/S

La tabla siguiente describe los valores de error EGL para las operaciones de entrada/salida (E/S) que afectan a bases de datos, archivos y colas de mensajes MQSeries. Los valores asociados con errores graves están a disposición del código sólo si la variable de sistema `VGVar.handleHardIOErrors` se ha establecido en 1, según se describe en el apartado *Manejo de excepciones*.

Valor de error	Tipo de error	Tipo de registro	Significado del valor de error
deadLock	Grave	SQL	Dos instancias de programa están intentado cambiar un registro, pero ninguna de ellas puede hacerlo sin la intervención del sistema.
duplicate	Leve	Indexado o relativo	El código ha intentado acceder a un registro con una clave que ya existe, y el intento ha sido satisfactorio. Para obtener detalles, consulte <i>duplicate</i> .
endOfFile	Leve	Indexado, relativo, serie	Para obtener detalles, consulte <i>endOfFile</i> .

Valor de error	Tipo de error	Tipo de registro	Significado del valor de error
ioError	Grave o leve	Cualquiera	EGL ha recibido un código de retorno no cero de la operación de E/S.
format	Grave	Cualquiera	El archivo al que se ha accedido es incompatible con la definición de registro. Para obtener detalles, consulte <i>format</i> .
fileNotAvailable	Grave	Cualquiera	El valor fileNotAvailable es posible para cualquier operación de E/S y puede indicar, por ejemplo, que otro programa está utilizando el archivo o que los recursos necesarios para acceder al archivo son insuficientes.
fileNotFound	Grave	Indexado, cola de mensajes, relativo, serie	No se ha encontrado un archivo.
full	Grave	Indexado, relativo, serie	El valor full se establece en estos casos: <ul style="list-style-type: none"> <li>• Un archivo indexado o serie está lleno</li> </ul>
hardIOError	Grave	Cualquiera	Se ha producido un error grave, que puede ser cualquiera excepto endOfFile, noRecordFound o duplicate.
noRecordFound	Leve	Cualquiera	Para obtener detalles, consulte <i>noRecordFound</i> .
unique	Grave	Indexado, relativo o SQL	UNQ indica <i>exclusivo</i> : el código ha intentado añadir o sustituir un registro con una clave que ya existe, y el intento ha fallado. Para obtener detalles, consulte <i>unique</i> .

## duplicate

Para un registro indexado o relativo, **duplicate** se establece en estos casos:

- Una sentencia **add** intenta insertar un registro cuya clave o ID de registro ya existe en el archivo o en un índice alternativo, y la inserción es satisfactoria.
- Una sentencia **replace** sobrescribe un registro satisfactoriamente, y los valores de sustitución incluyen una clave que es la misma que la clave de índice alternativo de otro registro.
- Una sentencia **get**, **get next** o **get previous** lee un registro satisfactoriamente (o una sentencia **set** del formato *set record position* se ejecuta satisfactoriamente) y un segundo registro tiene la misma clave.

El valor **duplicate** sólo se devuelve si el método de acceso devuelve la información, al igual que en algunos sistemas operativos, pero no en todos. La opción no está disponible durante durante el acceso a bases de datos SQL.

## endOfFile

**endOfFile** se establece en estas condiciones:

- El código emite una sentencia **get next** para un registro serie o relativo cuando el puntero del archivo relacionado está al final del archivo. El puntero está al final cuando una sentencia **get** o **get next** anterior ha accedido al último registro del archivo.

- El código emite una sentencia **get next** para un registro indexado cuando el puntero del archivo relacionado está al final del archivo, como ocurre en estas situaciones:
  - Una sentencia **get** o **get next** anterior ha accedido al último registro del archivo; o
  - Una sentencia **set** de tipo *set record position* anterior ha accedido al último registro del archivo mientras se producía una de las siguientes situaciones:
    - El valor de clave coincidía con la clave del último registro del archivo; o
    - Todos los bytes del valor de clave estaban establecidos en FF hexadecimal. (Si una sentencia **set** de tipo *set record position* se ejecuta con un valor de clave establecido en FF hexadecimal, la sentencia establece la posición del puntero al final del archivo).
- El código emite una sentencia **get previous** para un registro indexado cuando el puntero del archivo relacionado está al principio del archivo, como ocurre en estas situaciones:
  - Una sentencia **get** o **get previous** anterior ha accedido al primer registro del archivo;
  - El código no ha accedido anteriormente al mismo archivo; o
  - Una sentencia **set** de tipo *set record position* se ha ejecutado con una clave cuando en el archivo no había claves anteriores a esa clave.
- Una sentencia **get next** intenta recuperar datos de un archivo vacío o no inicializado en un registro indexado.  
(Un archivo vacío es aquél del que se han suprimido todos los registros. Un archivo no inicializado es aquél al que nunca se han añadido registros).
- Una sentencia **get previous** intenta recuperar datos de un archivo vacío o no inicializado en un registro indexado.

## format

**format** puede resultar de cualquier tipo de operación de E/S y puede establecerse por las siguientes razones, entre otras:

- **Formato de registro**  
El formato de archivo (longitud fija o variable) es diferente del formato de registro EGL.
- **Longitud de registro**  
En relación a los registros de longitud fija, la longitud de un registro del archivo es diferente de la longitud del registro EGL. En relación a los registros de longitud variable, la longitud de un registro del archivo es mayor que la longitud del registro EGL.
- **Tipo de archivo**  
El tipo de archivo especificado para el registro no coincide con el tipo de archivo durante la ejecución.
- **Longitud de clave**  
La longitud de clave del archivo es diferente de la longitud de clave del registro indexado EGL.
- **Desplazamiento de clave**  
La posición de la clave en el archivo es diferente de la posición de la clave en el registro indexado EGL.

## noRecordFound

**noRecordFound** se establece en estas condiciones:

- Para un registro indexado, no se encuentra ningún registro que coincida con la clave especificada en una sentencia **get**.
- Para Java generado por EGL, el código emite una sentencia **get next** o **get previous** para un registro indexado cuando el archivo VSAM está vacío o no se ha iniciado.
- Para un registro relativo, no se encuentra ningún registro que coincida con el ID de registro especificado en una sentencia **get**. Como alternativa, una sentencia **get next** intenta acceder a un registro que está más allá del final del archivo.
- Para un registro SQL, no se encuentra ninguna fila que coincida con la sentencia SELECT especificada, o se produce una sentencia **get next** cuando no quedan filas seleccionadas para revisar.

## unique

Para un registro indexado o relativo, **unique** se establece en estos casos:

- Una sentencia **add** intenta insertar un registro cuya clave o ID de registro ya existe en el archivo o en un índice alternativo, y la inserción falla debido a la duplicación.
- Una sentencia **replace** no puede sobrescribir un registro debido a que los valores de sustitución incluyen una clave que es la misma que la clave de índice alternativo de otro registro.

El valor **unique** sólo se devuelve si el método de acceso devuelve la información, al igual que en algunos sistemas operativos, pero no en todos.

Durante el acceso a bases de datos SQL, **unique** se establece cuando un fila SQL que se añade o sustituye tiene una clave que ya existe en un índice exclusivo. El sqlcode correspondiente es -803.

### Consulta relacionada

“add” en la página 561

“Elementos de asociación” en la página 375

“close” en la página 568

“delete” en la página 571

“Manejo de excepciones” en la página 94

“execute” en la página 574

“get” en la página 585

“get next” en la página 597

“get previous” en la página 602

“Expresiones lógicas” en la página 497

“open” en la página 616

“prepare” en la página 630

“replace” en la página 632

---

## Operador isa

El operador **isa** es un operador binario que prueba si una expresión es de un tipo determinado. El objetivo principal consiste en probar el tipo de datos contenido en un campo de tipo ANY.

El operador se utiliza en una expresión lógica elemental con el formato siguiente:

*expresiónPrueba isa especificaciónTipo*

### *expresiónPrueba*

Una expresión numérica, de texto o de fecha y hora que puede estar compuesta de un solo campo o literal.

### *especificaciónTipo*

Una especificación de tipo que puede ser uno de los siguientes:

- Un nombre de componente.
- Una especificación de tipo primitivo como por ejemplo STRING; sin embargo, si el tipo primitivo puede asociarse a una longitud, la longitud debe especificarse como en estos ejemplos:

- BIN(9)
- CHAR(5)

No incluya una máscara de fecha y hora.

- Una especificación de tipo (tal como se ha descrito anteriormente) seguida por corchetes emparejados. En este caso, la especificación completa indica una matriz dinámica de un tipo determinado, longitud (donde proceda) y número de dimensiones.

La expresión lógica se resuelve en true si *expresiónPrueba* coincide con el tipo identificado en *especificaciónTipo* y de lo contrario se resuelve en false.

### **Consulta relacionada**

“Matrices” en la página 74

“Expresiones lógicas” en la página 497

“Operadores y precedencia” en la página 673

---

## **Propiedades de ejecución de Java (detalles)**

La tabla siguiente describe las propiedades que pueden incluirse en el descriptor de despliegue o en el archivo de propiedades del programa, así como el origen del valor generado en el archivo de entorno J2EE, si lo hay. El tipo Java para cada propiedad es java.lang.String a menos que la columna de descripción indique lo contrario.

<b>Propiedad de tiempo de ejecución</b>	<b>Descripción</b>	<b>Origen del valor generado</b>
cso.cicsj2c.timeout	<p>Especifica el número de milisegundos antes de que se exceda el tiempo de espera durante una llamada que utilice el protocolo CICSJ2C. El valor por omisión es 30000, que representa 30 segundos. Si el valor se establece en 0, no se produce un tiempo de espera excedido. El valor debe ser 0 o mayor.</p> <p>El tipo Java en este caso es Java.lang.Integer.</p> <p>La propiedad no tiene efecto alguno sobre las llamadas cuando el código se ejecuta en WebSphere 390; encontrará los detalles en <i>Configuración del servidor J2EE para llamadas CICSJ2C</i>.</p>	<p>Opción del descriptor de construcción <b>cicsj2cTimeout</b></p>

Propiedad de tiempo de ejecución	Descripción	Origen del valor generado
cso.linkageOptions.LO	Especifica el nombre de un archivo de propiedades de enlace que guía la forma en que el programa generado o la envoltura llama a otros programas. <i>LO</i> es el nombre del componente de opciones de enlace utilizado en la generación. Encontrará los detalles en <i>Desplegar un archivo de propiedades de enlace</i> .	<i>LO</i> es de la opción del descriptor de construcción <b>linkage</b> y el valor predeterminado es el nombre del componente de opciones de enlace seguido por la extensión <i>.properties</i>
tcpiplistener.port	Especifica el número del puerto en el que está a la escucha un escucha de TCP/IP de EGL (de la clase CSOTcpipListener o CSOTcpipListenerJ2EE). No existe un valor por omisión. Para conocer más detalles, consulte los temas que hace referencia a <i>Configuración de la escucha de TCP/IP</i> .  El tipo Java en este caso es Java.lang.Integer.	No generado
tcpiplistener.trace.file	Especifica el nombre del archivo en que se registrará la actividad de uno o varios escuchas de TCP/IP de EGL (todos son de la clase CSOTcpipListener o CSOTcpipListenerJ2EE). El archivo por omisión es tcpiplistener.out.	No generado; el rastreo es para uso exclusivo de IBM
tcpiplistener.trace.flag	Especifica si debe rastrearse la actividad de uno o varios escuchas de TCP/IP de EGL (todos de la clase CSOTcpipListener o CSOTcpipListenerJ2EE). Seleccione una de estas opciones: <ul style="list-style-type: none"> <li>• 1 para registrar la actividad en el archivo identificado en la propiedad <b>tcpiplistener.trace.flag</b></li> <li>• 0 (el valor por omisión) para no registrar la actividad</li> </ul> El tipo Java en este caso es Java.lang.Integer. Para conocer más detalles, consulte los temas que hace referencia a <i>Configuración de la escucha de TCP/IP</i> .	No generado; el rastreo es para uso exclusivo de IBM

Propiedad de tiempo de ejecución	Descripción	Origen del valor generado
vgj.datemask. gregorian.long.locale	<p>Contiene la máscara de fecha utilizada en cualquiera de dos casos:</p> <ul style="list-style-type: none"> <li>• Se invoca el código Java generado para la variable del sistema <code>VGVar.currentFormattedGregorianCalendar</code>, o bien</li> <li>• EGL valida un campo de formulario de texto o elemento de página que tiene una longitud de 10 o más, si la propiedad de elemento <b>dateFormat</b> está establecida en <code>formatoFechaGregorianoSistema</code>.</li> </ul> <p><i>locale</i> es el código especificado en la propiedad <b>vgj.nls.code</b>. En las aplicaciones Web, puede cambiar la propiedad de máscara de fecha en uso asignando un valor distinto a <code>sysLib.setLocale</code>.</p>	El valor del descriptor de construcción para la máscara de fecha Gregoriana larga; el valor por omisión es específico del entorno local
vgj.datemask. gregorian.short.locale	<p>Contiene la máscara de fecha utilizada cuando EGL valida un elemento de página o un campo de formulario de texto que tiene una longitud de menos de 10, si la propiedad del elemento <b>dateFormat</b> está establecida como <code>formatoFechaGregorianoSistema</code>.</p> <p><i>locale</i> es el código especificado en la propiedad <b>vgj.nls.code</b>. En las aplicaciones Web, puede cambiar la propiedad de máscara de fecha en uso asignando un valor distinto a <code>sysLib.setLocale</code>.</p>	El valor del descriptor de construcción para la máscara de fecha Gregoriana corta; el valor por omisión es específico del entorno local
vgj.datemask. julian.long.locale	<p>Contiene la máscara de fecha utilizada en cualquiera de dos casos:</p> <ul style="list-style-type: none"> <li>• Se invoca el código Java generado para la variable de sistema <code>VGVar.currentFormattedJulianDate</code>, o bien</li> <li>• EGL valida un campo de formulario de texto o elemento de página que tiene una longitud de 10 o más, si la propiedad de elemento <b>dateFormat</b> está establecida en <code>formatoFechaJulianoSistema</code>.</li> </ul> <p><i>locale</i> es el código especificado en la propiedad <b>vgj.nls.code</b>. En las aplicaciones Web, puede cambiar la propiedad de máscara de fecha en uso asignando un valor distinto a <code>sysLib.setLocale</code>.</p>	El valor del descriptor de construcción para la máscara de fecha Juliana larga; el valor por omisión es específico del entorno local

Propiedad de tiempo de ejecución	Descripción	Origen del valor generado
vgj.datemask. julian.short.locale	<p>Contiene la máscara de fecha utilizada cuando EGL valida un elemento de página o un campo de formulario de texto que tiene una longitud de menos de 10, si la propiedad del elemento <b>dateFormat</b> está establecida como <i>formatoFechaJulianoSistema</i>.</p> <p><i>locale</i> es el código especificado en la propiedad <b>vgj.nls.code</b>. En las aplicaciones Web, puede cambiar la propiedad de máscara de fecha en uso asignando un valor distinto a <code>sysLib.setLocale</code>.</p>	El valor del descriptor de construcción para la máscara de fecha Juliana corta; el valor por omisión es específico del entorno local
vgj.default.databaseDelimiter	Especifica el símbolo utilizado para separar un valor del siguiente en las funciones de sistema <b>SysLib.loadTable</b> y <b>SysLib.unLoadTable</b> . El valor por omisión es la barra vertical ( ).	
vgj.default.dateFormat	Establece el valor inicial de la variable de sistema <b>StrLib.defaultDateFormat</b> ; para obtener detalles acerca de los valores válidos, consulte el apartado <i>Especificadores de fecha, hora e indicación de la hora</i>	
vgj.defaultI4GLNativeLibrary	Especifica el nombre de DLL al que accede una biblioteca de tipo <code>nativeLibrary</code> . La propiedad es obligatoria si no ha especificado la propiedad de biblioteca <b>dllName</b>	
vgj.default.moneyFormat	Establece el valor inicial de la variable de sistema <b>StrLib.defaultMoneyFormat</b> ; para obtener detalles acerca de los valores válidos, consulte el apartado relativo a <i>formatNumber()</i> .	
vgj.default.numericFormat	Establece el valor inicial de la variable de sistema <b>StrLib.defaultNumericFormat</b> ; para obtener detalles acerca de los valores válidos, consulte el apartado relativo a <i>formatNumber()</i> .	
vgj.default.timeFormat	Establece el valor inicial de la variable de sistema <b>StrLib.defaultTimeFormat</b> ; para obtener detalles acerca de los valores válidos, consulte el apartado <i>Especificadores de fecha, hora e indicación de la hora</i>	

Propiedad de tiempo de ejecución	Descripción	Origen del valor generado
vgj.default.timestampFormat	Establece el valor inicial de la variable de sistema <b>StrLib.defaultTimestampFormat</b> ; para obtener detalles acerca de los valores válidos, consulte el apartado <i>Especificadores de fecha, hora e indicación de la hora</i>	
vgj.jdbc.database.SN	<p>Especifica el nombre de base de datos JDBC que se utiliza cuando se realiza una conexión a base de datos mediante la función del sistema <code>sysLib.connect</code> o <code>VGLib.connectionService</code>.</p> <p>El significado del valor es distinto para las conexiones J2EE comparado con las conexiones estándar (no J2EE):</p> <ul style="list-style-type: none"> <li>• En relación con las conexiones J2EE (necesario en un entorno de producción), el valor es el nombre al que se enlaza el origen de datos en el registro de JNDI; por ejemplo, <code>jdbc/MyDB</code></li> <li>• En relación con una conexión JDBC estándar (como la que podría utilizarse para depurar), el valor es la URL de conexión; por ejemplo, <code>jdbc:db2:MyDB</code></li> </ul> <p>Debe personalizar el nombre de la propiedad al especificar un valor de sustitución para <i>SN</i>, en el momento del despliegue. El valor de sustitución debe coincidir a su vez con el nombre de servidor incluido en la invocación de <code>VGLib.connectionService</code> o el nombre de base de datos incluido en la invocación de <code>sysLib.connect</code>.</p>	El valor del descriptor de construcción para el nombre de base de datos que desea asociar con el "nombre de servidor" especificado
vgj.jdbc.default.database.autoCommit	Especifica si se produce un compromiso después de cada cambio en la base de datos por omisión. Los valores válidos son <code>true</code> y <code>false</code> , tal como se describe en el apartado <i>sqlCommitControl</i> .	Opción de descriptor de construcción <b>sqlCommitControl</b>

Propiedad de tiempo de ejecución	Descripción	Origen del valor generado
vgj.jdbc.default.database. <i>programName</i>	<p>Especifica el nombre de base de datos por omisión que se utiliza para una operación de E/S de SQL si no existe una conexión a base de datos previa. EGL incluye el nombre de programa (o alias de programa, si lo hay) como un valor de sustitución para <i>programName</i> de forma que cada programa tenga su propia base de datos por omisión. El nombre de programa es opcional, no obstante, y se utiliza una propiedad denominada <code>vgj.jdbc.default.database</code> como valor por omisión para cualquier programa no referenciado en una propiedad específica de programa de esta clase.</p> <p>El significado del valor de la propiedad es distinto para las conexiones J2EE en comparación con las conexiones no J2EE:</p> <ul style="list-style-type: none"> <li>• En relación con las conexiones J2EE, el valor es el nombre al que se enlaza el origen de datos en el registro de JNDI; por ejemplo, <code>jdbc/MyDB</code></li> <li>• En relación con una conexión JDBC estándar, el valor es la URL de conexión; por ejemplo, <code>jdbc:db2:MyDB</code></li> </ul>	<p>Dependen del tipo de conexión:</p> <ul style="list-style-type: none"> <li>• Para conexiones J2EE, la opción del descriptor de construcción <b>sqlJNDIName</b></li> <li>• Para conexiones no J2EE, la opción del descriptor de construcción <b>sqlIDB</b></li> </ul>
vgj.jdbc.default.password	<p>Especifica la contraseña para acceder a la conexión de base de datos identificada en <b>vgj.jdbc.default.database</b>.</p> <p>Para evitar exponer las contraseñas en el archivo de entorno J2EE, realice una de las siguientes tareas:</p> <ul style="list-style-type: none"> <li>• Especifique una contraseña en scripts de programa y función utilizando la función del sistema <code>sysLib.connect</code> o <code>VGLib.connectionService</code>; o bien</li> <li>• Incluya un ID de usuario y contraseña en la especificación de origen de datos en el servidor de aplicaciones web, tal como se describe en <i>Configuración de una conexión JDBC J2EE</i>.</li> </ul>	<p>Opción del descriptor de construcción <b>sqlPassword</b></p>
vgj.jdbc.default.userid	<p>Especifica el ID de usuario para acceder a la conexión a base de datos identificada en <b>vgj.jdbc.default.database</b>.</p>	<p>Opción del descriptor de construcción <b>sqlID</b></p>

Propiedad de tiempo de ejecución	Descripción	Origen del valor generado
vgj.jdbc.drivers	Especifica la clase de controlador para acceder a la conexión a base de datos identificada en <b>vgj.jdbc.default.database</b> . Esta propiedad no está presente en el descriptor de despliegue o el archivo de entorno J2EE y se utiliza solamente para una conexión JDBC estándar (no J2EE).	Opción del descriptor de construcción <b>sqlJDBCDriverClass</b>
vgj.messages.file	Especifica un archivo de propiedades que incluye mensajes creados o personalizados. Se busca en el archivo en los siguientes casos: <ul style="list-style-type: none"> <li>• Cuando el tiempo de ejecución de EGL responde a la invocación de la función <code>SysLib.getMessage</code> que devuelve un mensaje creado; para conocer más detalles, consulte el apartado <i>SysLib.getMessage</i></li> <li>• Cuando el tiempo de ejecución de EGL maneja una aplicación <code>consoleUI</code> e intenta presentar texto de comentario o ayuda de un archivo identificado en la variable de sistema <b>ConsoleLib.messageResource</b>, pero dicha variable no tiene ningún valor.</li> <li>• Cuando EGL intenta visualizar un mensaje de tiempo de ejecución Java, tal como se explica en <i>Personalización de mensajes para mensajes de tiempo de ejecución de EGL</i></li> </ul>	
vgj.nls.code	Especifica el código NLS de tres letras del programa. Para obtener una lista de valores válidos, consulte <code>targetNLS</code> .  Si la propiedad no está establecida, se aplican estas normas: <ul style="list-style-type: none"> <li>• El valor toma por omisión el código NLS que corresponde al entorno local de Java por omisión</li> <li>• El valor es <code>ENU</code> si el entorno local de Java por omisión no corresponde a ninguno de los códigos NLS soportados por EGL</li> </ul>	Opción del descriptor de construcción <b>targetNLS</b>
vgj.nls.currency	Especifica el carácter utilizado como símbolo de moneda. El valor por omisión está determinado por el entorno local asociado con <b>vgj.nls.code</b> .	Opción del descriptor de construcción <b>currencySymbol</b>

Propiedad de tiempo de ejecución	Descripción	Origen del valor generado
vgj.nls.number.decimal	<p>Especifica el carácter utilizado como símbolo decimal. El valor por omisión está determinado por el entorno local asociado con <b>vgj.nls.code</b>.</p>	Opción del descriptor de construcción <b>decimalSymbol</b>
vgj.properties.file	<p>Sólo se utiliza si el primer programa de una unidad de ejecución no J2EE se generó con VisualAge Generator o con una versión de EGL anterior a 6.0.</p> <p><b>vgj.properties.file</b> especifica un archivo de propiedades alternativo. El archivo se utiliza en una unidad de ejecución no J2EE en lugar de cualquier archivo de propiedades de programa no global. La utilización del archivo global no se ve afectada. (En unidades de ejecución cuyo primer programa se generó con el EGL antiguo o con VisualAge Generator, el archivo global se llama <b>vgj.properties</b>.)</p> <p>El archivo al que hace referencia la propiedad <b>vgj.properties.file</b> solo se utiliza si incluye esa propiedad en una directiva de línea de mandatos, como en este ejemplo:</p> <pre>java -Dvgj.properties.file=c:\new.properties</pre> <p>El valor de <b>vgj.properties.file</b> incluye la vía de acceso totalmente calificada al archivo de propiedades.</p> <p>El hecho de especificar la propiedad <b>vgj.properties.file</b> en un archivo de propiedades no surte efecto.</p>	
vgj.ra.QN.conversionTable	<p>Especifica el nombre de la tabla de conversión utilizada por un programa Java generado durante el acceso de la cola de mensajes MQSeries identificada por <i>QN</i>. Los valores válidos son programControlled, NONE, o un nombre de tabla de conversión. El valor por omisión es NONE.</p>	Propiedad de asociaciones de recurso <b>conversionTable</b>

Propiedad de tiempo de ejecución	Descripción	Origen del valor generado
vgj.ra.FN.fileType	<p>Especifica el tipo de archivo asociado con <i>FN</i>, que es un archivo o nombre de cola identificado en el componente de registro. El valor de la propiedad es <i>seqws</i> o <i>mq</i>, tal como se describe en <i>Referencia cruzada de tipos de registro y archivo</i>.</p> <p>Debe especificar esta propiedad del descriptor de despliegue para cada archivo lógico que utilice el programa.</p>	Propiedad de asociaciones de recurso <b>fileType</b>
vgj.ra.FN.replace	<p>Especifica el efecto de una sentencia <i>add</i> en un registro asociado con <i>FN</i>, que es un nombre de archivo identificado en un registro. Seleccione uno de dos valores:</p> <ul style="list-style-type: none"> <li>• 1 si la sentencia sustituye al registro de archivo</li> <li>• 0 (el valor por omisión) si la sentencia añade un registro al archivo</li> </ul> <p>El tipo Java en este caso es <code>java.lang.Integer</code>.</p>	Propiedad de asociaciones de recurso <b>replace</b>
vgj.ra.FN.systemName	<p>Especifica el nombre del archivo físico o cola de mensajes asociado con <i>FN</i>, que es un archivo o nombre de cola identificado en el componente de registro.</p> <p>Debe especificar esta propiedad del descriptor de despliegue para cada archivo lógico que utilice el programa.</p>	Propiedad de asociaciones de recurso <b>systemName</b>

Propiedad de tiempo de ejecución	Descripción	Origen del valor generado
vgj.ra.FN.text	<p>Especifica si debe provocarse que un programa Java generado haga lo siguiente al acceder a un archivo por medio de un registro serie:</p> <ul style="list-style-type: none"> <li>• Añadir caracteres de fin de la línea durante la operación <b>añadir</b>. En plataformas no UNIX, esos caracteres son los de retorno de carro y salto de línea; en plataformas UNIX, el único carácter es es salto de línea.</li> <li>• Eliminar caracteres de fin de la línea durante la operación <b>obtener siguiente</b>.</li> </ul> <p><i>FN</i> es el nombre de archivo asociado con el registro de serie.</p> <p>Seleccione uno de estos valores:</p> <ul style="list-style-type: none"> <li>• 1 para realizar los cambios</li> <li>• 0 (el valor por omisión) para no realizar los cambios</li> </ul> <p>El tipo Java en este caso es <code>java.lang.Integer</code>.</p>	Propiedad de asociaciones de recurso <b>text</b>
vgj.trace.device.option	<p>Destino de los datos de rastreo, si los hay. Seleccione uno de estos valores:</p> <ul style="list-style-type: none"> <li>• 0 para grabar en <code>System.out</code></li> <li>• 1 para grabar en <code>System.err</code></li> <li>• 2 (el valor por omisión) para grabar en el archivo especificado en <b>vgj.trace.device.spec</b> con esta excepción: para rastreos de E/S de VSAM, grabe en <code>vsam.out</code></li> </ul> <p>El tipo Java en este caso es <code>java.lang.Integer</code>.</p>	El valor generado, si lo hay, es 2
vgj.trace.device.spec	<p>Especifica el nombre del archivo de salida si <b>vgj.trace.device.option</b> está establecido en 2. La excepción es que los rastreos de E/S de VSAM se graban en <code>vsam.out</code>.</p>	El valor generado, si lo hay, es <code>vgjtrace.out</code>

Propiedad de tiempo de ejecución	Descripción	Origen del valor generado
vgj.trace.type	<p>Especifica el valor de rastreo de ejecución. Suma los valores de interés para obtener el rastreo deseado:</p> <ul style="list-style-type: none"> <li>• -1 para rastrear todo</li> <li>• 0 para ningún rastreo (el valor por omisión)</li> <li>• 1 para rastreo general, incluidas invocaciones de funciones y sentencias call</li> <li>• 2 para funciones del sistema que manejan matemáticas</li> <li>• 4 para funciones del sistema que manejan series</li> <li>• 16 para datos pasados en una sentencia call</li> <li>• 32 para las opciones de enlace utilizadas en una llamada</li> <li>• 128 para E/S de jdbc</li> <li>• 256 para E/S de archivo</li> <li>• 512 para todas las propiedades excepto vgj.jdbc.default.password</li> </ul> <p>El tipo Java en este caso es <code>java.lang.Integer</code>.</p>	El valor generado, si lo hay, es 0

### Conceptos relacionados

- “Propiedades de tiempo de ejecución Java” en la página 347
- “Componente de biblioteca de tipo `basicLibrary`” en la página 142
- “Archivo de propiedades de enlace” en la página 364

### Tareas relacionadas

- “Desplegar un archivo de propiedades de enlace” en la página 364
- “Establecer una conexión JDBC J2EE” en la página 362
- “Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354
- “Config. escucha TCP/IP para aplic. llamada en módulo cliente aplic. J2EE” en la página 359
- “Configurar el escucha TCP/IP para una aplicación no J2EE llamada” en la página 353
- “Cómo se realiza una conexión JDBC estándar” en la página 263

### Consulta relacionada

- “Elemento `callLink`” en la página 407
- “`cicsj2cTimeout`” en la página 386
- “`connect()`” en la página 895
- “`connectionService()`” en la página 916
- “`currentFormattedGregorianCalendar`” en la página 941
- “`currentFormattedJulianDate`” en la página 942
- “`currentShortGregorianCalendar`” en la página 945
- “`currentShortJulianDate`” en la página 945
- “Especificadores de fecha, hora e indicación de la hora” en la página 45
- “`decimalSymbol`” en la página 387
- “`defaultDateFormat`” en la página 874

“defaultMoneyFormat” en la página 875  
“defaultNumericFormat” en la página 875  
“defaultTimeFormat” en la página 875  
“defaultTimestampFormat” en la página 876  
“formatNumber()” en la página 878  
“getMessage()” en la página 902  
“linkage” en la página 397  
“Archivo de propiedades de enlace (detalles)” en la página 657  
“loadTable()” en la página 904  
“Personalización de mensajes para el tiempo de ejecución de Java EGL” en la página 661  
“Referencias cruzadas de tipo de registro y tipo de archivo” en la página 737  
“setLocale()” en la página 907  
“sqlCommitControl” en la página 400  
“sqlDB” en la página 400  
“sqlID” en la página 401  
“sqlJDBCClass” en la página 401  
“sqlJNDIName” en la página 402  
“sqlPassword” en la página 403  
“targetNLS” en la página 405  
“unloadTable()” en la página 912

---

## Clases de envoltura Java

Al solicitar que un componente de programa se genere como envoltura Java, EGL produce una clase de envoltura para cada uno de los siguientes elementos:

- El programa generado
- Cada registro fijo o formulario que se declara como parámetro en ese programa
- Cada matriz dinámica que se declara como parámetro; si la matriz es de registros fijos, la clase de la matriz dinámica se añade a la clase del propio registro fijo
- Cada elemento de estructura que tenga estas características:
  - Es uno de los registros fijos o formularios para los que se genera una clase de envoltura
  - Tiene como mínimo un elemento de estructura subordinado; en otras palabras, está subestructurado
  - Es una matriz; en este caso, una matriz subestructurada

A continuación se ofrece un ejemplo de componente de registro fijo con una matriz subestructurada:

```
Record myPart type basicRecord
 10 MyTopStructure CHAR(20) [5];
 20 MyStructureItem01 CHAR(10);
 20 MyStructureItem02 CHAR(10);
end
```

Las descripciones subsiguientes hacen referencia a las clases de envoltura de un programa determinado como *clase de envoltura de programa*, *clases de envoltura de parámetro*, *clases de envoltura de matriz dinámica* y *clases de envoltura de matriz de elementos subestructurada*.

EGL genera una clase BeanInfo para cada clase de envoltura de parámetro, clase de envoltura de matriz dinámica o clase de envoltura de matriz de elementos subestructurada. La clase BeanInfo permite utilizar la clase de envoltura relacionada como bean Java compatible con Java. Probablemente, no interactuará con la clase BeanInfo.

Cuando genera una envoltura, la lista de parámetros del programa llamado no puede incluir parámetros de tipo BLOB, CLOB, STRING, Dictionary, ArrayDictionary o registro no fijo.

## Visión general de la utilización de las clases de envoltura

Para utilizar las clases de envoltura para comunicarse con un programa generado con VisualAge Generator, haga lo siguiente en el programa nativo Java:

- Cree una instancia de una clase (una subclase de CSOPowerServer) para suministrar servicios de middleware, como por ejemplo conversión de datos entre el código nativo Java y un programa generado:

```
import com.ibm.javart.v6.cso.*;

public class MyNativeClass
{
 /* declarar una variable para middleware */
 CSOPowerServer powerServer = null;

 try
 {
 powerServer = new CSOLocalPowerServerProxy();
 }
 catch (CSOException exception)
 {
 System.out.println("Error al inicializar el middleware"
 + exception.getMessage());
 System.exit(8);
 }
}
```

- Cree una instancia de una clase de envoltura de programa para hacer lo siguiente:
  - Asignar estructuras de datos, incluidas las matrices dinámicas si las hay
  - Proporcionar acceso a los métodos que, a su vez, acceden al programa generado

La llamada al constructor incluye el objeto de middleware:

```
myProgram = new MyprogramWrapper(powerServer);
```

- Declare las variables que se basan en las clases de envoltura de parámetro:

```
Mypart myParm = myProgram.getMyParm();
Mypart2 myParm2 = myProgram.getMyParm2();
```

Si el programa contiene parámetros que son matrices dinámicas, declare variables adicionales cada una de las cuales se base en una clase de envoltura de matriz dinámica:

```
myRecArrayVar myParm3 = myProgram.getMyParm3();
```

Para obtener detalles acerca de cómo interactuar con matrices dinámicas, consulte el apartado “Clases de envoltura de matriz dinámica” en la página 557.

- En la mayoría de los casos, (como en el paso anterior), utilizará las variables de parámetro para hacer referencia y cambiar la memoria asignada en el objeto de envoltura de programa
- Defina un ID de usuario y contraseña, pero solamente en estos casos:
  - La envoltura Java accede a un programa basado en iSeries mediante iSeries Toolbox para Java; o bien
  - El programa generado se ejecuta en una región CICS para z/OS que autentica el acceso remoto.

El ID de usuario y la contraseña no se utilizan para el acceso a bases de datos.

El ID de usuario y la contraseña se establecen y revisan mediante la variable `callOptions` del objeto programa, como en este ejemplo:

```
myProgram.callOptions.setUserID("myID");
myProgram.callOptions.setPassword("myWord");
myUserID = myProgram.callOptions.getUserID();
myPassword = myProgram.callOptions.getPassword();
```

- Acceda al programa generado, en la mayoría de los casos invocando el método `execute` del objeto de envoltura de programa:

```
myProgram.execute();
```
- Utilice el objeto de middleware para establecer el control de transacción de base de datos, pero sólo en la siguiente situación:
  - El objeto de envoltura de programa accede a un programa generado en CICS para z/OS o accede a un programa COBOL basado en iSeries mediante IBM Toolbox para Java. En el segundo caso, el valor **remoteComType** para la llamada es `JAVA400`.
  - En el componente de opciones de enlace utilizado para generar las clases de envoltura, ha especificado que la unidad de trabajo de base de datos está bajo el control del cliente (en este caso, la envoltura); para obtener detalles, consulte la referencia a **luwControl** en *elemento callLink*.

Si la unidad de trabajo de base de datos está bajo el control del cliente, el proceso incluye la utilización de métodos de compromiso y retroacción del objeto de middleware:

```
powerServer.commit();
powerServer.rollback();
```

- Cierre el objeto de middleware y permita la recogida de basura:

```
if (powerServer != null)
{
 try
 {
 powerServer.close();
 powerServer = null;
 }

 catch(CSOException error)
 {
 System.out.println("Error al cerrar el middleware"
 + error.getMessage());
 System.exit(8);
 }
}
```

## La clase de envoltura de programa

la clase de envoltura de programa incluye una variable de instancia privada para cada parámetro del programa generado. Si el parámetro es un registro o un formulario, la variable hace referencia a una instancia de la clase de envoltura de parámetro relacionada. Si el parámetro es un elemento de datos, la variable tiene un tipo primitivo Java.

Al final de esta página de la ayuda figura una tabla que describe las conversiones entre EGL y los tipos Java.

Un objeto de envoltura de programa incluye los siguientes métodos públicos:

- Métodos **get** y **set** para cada parámetro, donde el formato del nombre es el siguiente:

```
finalidadnombreParámetro()
```

### *finalidad*

La palabra **get** o **set**

### *nombreParámetro*

Nombre del elemento de datos, registro o formulario; la primera letra es mayúscula, y el aspecto de las demás letras está determinado por el convenio de denominación descrito en el apartado “Convenios de denominación de las clases de envoltura Java” en la página 558

- Un método **execute** para llamar al programa; este método se utiliza si los datos que se pasarán como argumentos en la llamada están en la memoria asignada al objeto de envoltura de programa

En lugar de asignar valores a las variables de instancia, puede hacer lo siguiente:

- Asigne memoria a los objetos de envoltura de parámetro, objetos de envoltura de matriz dinámica y tipos primitivos
- Asigne valores a la memoria que ha asignado
- Pase esos valores al programa invocando el método **call** del objeto de envoltura de programa en lugar de invocar el método **execute**

El objeto de envoltura de programa también incluye la variable `callOptions`, que tiene las siguientes finalidades:

- Si ha generado la envoltura Java para que las opciones de enlace de la llamada se establezcan durante la generación, la variable `callOptions` contendrá la información de enlace. Para obtener detalles acerca de cuándo se establecen las opciones de enlace, consulte `remoteBind` en *elemento callLink*.
- Si ha generado la envoltura Java para que las opciones de enlace de la llamada se establezcan durante la ejecución, la variable `callOptions` contendrá el nombre del archivo de propiedades de enlace. El nombre del archivo es **LO.properties**, donde *LO* es el nombre del componente de opciones de enlace utilizado para la generación.
- En cualquier caso, la variable `callOptions` proporciona los siguientes métodos para establecer u obtener un ID de usuario y contraseña:

```
setPassword(contraseña)
setUserid(IDusuario)
getPassword()
getUserid()
```

El ID de usuario y contraseña se utilizan al establecer la propiedad **remoteComType** del elemento `callLink` en uno de los siguientes valores:

- CICSECI
- CICSJ2C
- JAVA400

Finalmente, considere la siguiente situación: el código nativo Java requiere notificación cuando se efectúa un cambio en un parámetro de tipo primitivo. Para hacer posible esta notificación, el código nativo se registra como escuchador invocando el método **addPropertyChangeListener** del objeto de envoltura de programa. En este caso, cualquiera de las siguientes situaciones desencadena el evento `PropertyChange` que hace que el código nativo reciba la notificación durante la ejecución:

- El código nativo invoca un método **set** en un parámetro de tipo primitivo
- El código generado devuelve datos cambiados a un parámetro de tipo primitivo

El evento `PropertyChange` se describe en la especificación `JavaBean` de Sun Microsystems, Inc.

## El conjunto de clases de envoltura de parámetro

Una clase de envoltura de parámetro se produce para cada registro declarado como parámetro en el programa generado. En el caso más común, una clase de envoltura de parámetro se utiliza sólo para declarar una variable que hace referencia al parámetro, como en el ejemplo siguiente:

```
Mypart myRecWrapperObject = myProgram.getMyrecord();
```

En este caso, se utiliza la memoria asignada por el objeto de envoltura de programa.

También puede utilizar la clase de envoltura de parámetro para declarar memoria, que es necesario si invoca al método `call` (en lugar de al método `execute`) del objeto programa.

La clase de envoltura de parámetro incluye un conjunto de variables de instancia privada, como las siguientes:

- Una variable de un tipo primitivo Java para cada de los elementos de estructura de bajo nivel del parámetro, pero sólo para los elementos de estructura que no sean matrices ni estén dentro de una matriz subestructurada
- Una matriz de tipo primitivo Java para cada elemento de estructura EGL que sea una matriz no subestructurada
- Un objeto de una clase de matriz interna para cada matriz subestructurada que no esté dentro de una matriz subestructurada; la clase interna puede tener clases internas anidadas para representar matrices subestructuradas subordinadas

La clase de envoltura de parámetro incluye varios métodos públicos:

- Un conjunto de métodos `get` y `set` permite obtener y establecer cada variable de instancia. El formato de cada nombre de método es el siguiente:

```
finalidadnombreEE()
```

*finalidad*

La palabra `get` o `set`.

*nombreEE*

Nombre del elemento de estructura. La primera letra es mayúscula, y el aspecto de las demás letras está determinado por el convenio de denominación descrito en el apartado "Convenios de denominación de las clases de envoltura Java" en la página 558.

**Nota:** Los elementos de estructura que declare como rellenos se incluirán en la llamada de programa; pero la clase de envoltura de matriz no incluyen métodos `get` y `set` públicos para dichos elementos de estructura.

- El método `equals` permite determinar si los valores almacenados en otro objeto de la misma clase son idénticos a los valores almacenados en el objeto de envoltura de parámetro. El método devuelve `true` sólo si las clases y valores son idénticos.
- El método `addPropertyChangeListener` se invoca si el programa requiere notificación de un cambio en una variable de un tipo primitivo Java.
- Un segundo conjunto de métodos `get` y `set` permite obtener y establecer los indicadores de nulo de cada elemento de estructura de un parámetro de registro SQL. El formato de cada uno de estos nombres de método es el siguiente:

```
finalidadnombreEENullIndicator()
```

*finalidad*

La palabra **get** o **set**.

*nombreEE*

Nombre del elemento de estructura. La primera letra es mayúscula, y el aspecto de las demás letras está determinado por el convenio de denominación descrito en el apartado “Convenios de denominación de las clases de envoltura Java” en la página 558.

## El conjunto de clases de envoltura de matriz de elementos subestructurada

Una clase de envoltura de matriz de elementos subestructurada es una clase interna de una clase de parámetro y representa una matriz subestructurada del parámetro relacionado. La clase de envoltura de matriz de elementos subestructurada incluye un conjunto de variables de instancia privada que hacen referencia a los elementos de estructura que se encuentran en la propia matriz y debajo de ella:

- Una variable de un tipo primitivo Java para cada uno de los elementos de estructura de bajo nivel de la matriz, pero sólo para los elementos de estructura que no sean matrices ni estén dentro de una matriz subestructurada
- Una matriz de tipo primitivo Java para cada elemento de estructura EGL que sea una matriz no subestructurada
- Un objeto de una clase de envoltura de matriz de elementos subestructurada interna para cada matriz subestructurada que no esté dentro de una matriz subestructurada; la clase interna puede tener clases internas anidadas para representar matrices subestructuradas subordinadas

La clase de envoltura de matriz de elementos subestructurada incluye los siguientes métodos:

- Un conjunto de métodos **get** y **set** para cada variable de instancia.

**Nota:** Los elementos de estructura que declare como rellenos se utilizarán en la llamada de programa; pero la clase de envoltura de matriz de elementos subestructurada no incluyen métodos **get** y **set** públicos para dichos elementos de estructura.

- El método **equals** permite determinar si los valores almacenados en otro objeto de la misma clase son idénticos a los valores almacenados en el objeto de envoltura de matriz de elementos subestructurada. El método devuelve **true** sólo si las clases y valores son idénticos.
- El método **addPropertyChangeListener** se invoca si el programa requiere notificación de un cambio en una variable de un tipo primitivo Java.

En la mayoría de los casos, el nombre de la clase de envoltura de matriz de elementos subestructurada de mayor nivel de una clase de envoltura de parámetro tiene el siguiente formato:

*nombreClaseParámetro.nombreClaseMatriz*

Observe el siguiente registro, por ejemplo:

```
Record CompanyPart type basicRecord
10 Departments CHAR(20)[5];
 20 CountryCode CHAR(10);
 20 FunctionCode CHAR(10)[3];
 30 FunctionCategory CHAR(4);
 30 FunctionDetail CHAR(6);
end
```

Si el parámetro `Company` se basa en `CompanyPart`, se utilizará la serie `CompanyPart.Departments` como el nombre de la clase interna.

Una clase interna de una clase interna amplía la utilización de una sintaxis con puntos. En este ejemplo, se utiliza el símbolo `CompanyPart.Departments.Functioncode` como el nombre de la clase interna de `Departments`.

Para obtener más detalles acerca de cómo denominar las clases de envoltura de matriz de elementos subestructurada, consulte el apartado *Salida de la generación de envoltura Java*.

## Clases de envoltura de matriz dinámica

Una clase de envoltura de matriz dinámica se produce para cada matriz dinámica declarada como parámetro en el programa generado. Observe la siguiente firma de programa EGL:

```
Program myProgram(intParms int[], recParms MyRec[])
```

El nombre de las clases de envoltura de matriz dinámica es `IntParmsArray` y `MyRecArray`.

Una clase de envoltura de matriz dinámica se utiliza para declarar una variable que hace referencia a la matriz dinámica, como en los ejemplos siguientes:

```
IntParmsArray myIntArrayVar = myProgram.getIntParms();
MyRecArray myRecArrayVar = myProgram.getRecParms();
```

Después de declarar las variables para cada matriz dinámica, puede añadir elementos:

```
// la adición a una matriz de tipos primitivos Java
// es un proceso de un solo paso
myIntArrayVar.add(new Integer(5));

// la adición a una matriz de registros o formularios
// requiere varios pasos; en este caso,
// empiece por asignar un objeto de registro nuevo
MyRec myLocalRec = (MyRec)myRecArrayVar.makeNewElement();

// los pasos para asignar valores no se muestran
// en este ejemplo, pero, después de asignar valores,
// añada el registro a la matriz
myRecArrayVar.add(myLocalRec);

// a continuación, ejecute el programa
myProgram.execute();

// cuando el programa efectúe el retorno, puede determinar
// el número de elementos de la matriz
int myIntArrayVarSize = myIntArrayVar.size();

// obtener el primer elemento de la matriz de enteros
// y convertirlo temporalmente a un objeto Integer
Integer firstIntElement = (Integer)myIntArrayVar.get(0);

// obtener el segundo elemento de la matriz de registros
// y convertirlo temporalmente a un objeto MyRec
MyRec secondRecElement = (MyRec)myRecArrayVar.get(1);
```

Como indica el ejemplo, EGL proporciona varios métodos para manipular las variables que haya declarado.

Método de la clase de matriz dinámica	Finalidad
<code>add(int, Objeto)</code>	Insertar un objeto en la posición especificada por <i>int</i> y desplazar el elemento actual y los sucesivos hacia la derecha.
<code>add(Objeto)</code>	Añadir un objeto al final de la matriz dinámica.
<code>addAll(ListaMatrices)</code>	Añadir un objeto ArrayList al final de la matriz dinámica.
<code>get()</code>	Recuperar el objeto ArrayList que contiene todos los elementos de la matriz.
<code>get(int)</code>	Recuperar el elemento que está en la posición especificada por <i>int</i>
<code>makeNewElement()</code>	Asignar un elemento nuevo del tipo específico de la matriz y recuperarlo sin añadirlo a la matriz dinámica.
<code>maxSize()</code>	Recuperar un entero que indica el número máximo (pero no real) de elementos de la matriz dinámica.
<code>remove(int)</code>	Eliminar el elemento que está en la posición especificada por <i>int</i>
<code>set(ArrayList)</code>	Utilizar el objeto ArrayList especificado como sustitución de la matriz dinámica.
<code>set(int, Objeto)</code>	Utilizar el objeto especificado como sustitución del elemento que está en la posición especificada por <i>int</i>
<code>size()</code>	Recuperar el número de elementos que se encuentran en la matriz dinámica.

En los siguientes casos, entre otros, se producen excepciones:

- Si especifica un índice no válido en el método **get** o **set**
- Si intenta añadir (o establecer) un elemento que pertenece a una clase incompatible con la clase de cada uno de los elementos de la matriz
- Si intenta añadir elementos a una matriz dinámica cuando el tamaño máximo de la matriz no puede dar soporte al aumento; y si el método **addAll** falla por esta razón, el método no añade elementos

## Convenios de denominación de las clases de envoltura Java

EGL crea los nombres de acuerdo con estas normas:

- Si el nombre está todo en mayúsculas, debe especificarlo todo en minúsculas.
- Si el nombre es una palabra clave, especifique un subrayado delante
- Si el nombre incluye un guión o un subrayado, elimine ese carácter y especifique la letra siguiente en mayúsculas
- Si el nombre incluye un signo de dólar (\$), un signo de arroba (@) o un signo de almohadilla (#), sustituya cada uno de esos caracteres por un doble subrayado (\_\_) y coloque un subrayado (\_) delante del nombre.
- Si el nombre se utiliza como nombre de clase, especifique la primera letra en mayúscula.

Las clases de envoltura de matriz dinámica están sujetas a las siguientes normas:

- En la mayoría de los casos, el nombre de una clase se basa en el nombre de la declaración de componente (elemento de datos, formulario o registro) que es la base de cada uno de los elementos de la matriz. Por ejemplo, si un componente

de registro se denomina MyRec y la declaración de matriz es recParms myRec[], la clase de envoltura de matriz dinámica relacionada se denominará MyRecArray.

- Si la matriz se basa en una declaración de elemento que no tiene ninguna declaración de componente relacionada, el nombre de la clase de matriz dinámica se basa en el nombre de la matriz. Por ejemplo, si la declaración de matriz es intParms int[], la clase de envoltura de matriz dinámica relacionada se denominará IntParmsArray.

## Referencias cruzadas de tipos de datos

La tabla siguiente indica la relación entre los tipos primitivos EGL del programa generado y los tipos de datos Java de la envoltura generada.

Tipo primitivo EGL	Longitud en caracteres o dígitos	Longitud en bytes	Decimales	Tipo de datos Java	Precisión máxima en Java
CHAR	1-32767	2-32766	NA	String	NA
DBCHAR	1-16383	1-32767	NA	String	NA
MBCHAR	1-32767	1-32767	NA	String	NA
UNICODE	1-16383	2-32766	NA	String	NA
HEX	2-75534	1-32767	NA	byte[]	NA
BIN, SMALLINT	4	2	0	short	4
BIN, INT	9	4	0	int	9
BIN, BIGINT	18	8	0	long	18
BIN	4	2	>0	float	4
BIN	9	4	>0	double	15
BIN	18	8	>0	double	15
DECIMAL, PACF	1-3	1-2	0	short	4
DECIMAL, PACF	4-9	3-5	0	int	9
DECIMAL, PACF	10-18	6-10	0	long	18
DECIMAL, PACF	1-5	1-3	>0	float	6
DECIMAL, PACF	7-18	4-10	>0	double	15
NUM, NUMC	1-4	1-4	0	short	4
NUM, NUMC	5-9	5-9	0	int	9
NUM, NUMC	10-18	10-18	0	long	18
NUM, NUMC	1-6	1-6	>0	float	6
NUM, NUMC	7-18	7-18	>0	double	15

### Conceptos relacionados

“Envoltura Java” en la página 301

“Configuraciones de tiempo de ejecución” en la página 9

### Tareas relacionadas

“Generar envolturas Java” en la página 300

### Consulta relacionada

“Elemento callLink” en la página 407

“Cómo se crean alias de los nombres de envoltura Java” en la página 670

“Archivo de propiedades de enlace (detalles)” en la página 657

“Salida de la generación de envoltura Java” en la página 676

“Propiedad remoteBind del elemento callLink” en la página 418

---

## Requisitos de controlador JDBC en EGL

Los requisitos de controlador JDBC varían según el sistema de gestión de base de datos, según sea para el tiempo de depuración o el tiempo de ejecución de EGL:

### DB2 UDB

El controlador DB2 Universal es compatible con EGL, pero el controlador de aplicaciones relacionado no es compatible; específicamente, el controlador de aplicaciones no puede procesar una sentencia EGL **open** o **get** que incluya la opción forUpdate.

IBM recomienda no utilizar en absoluto el controlador de red.

Si está utilizando aplicaciones J2EE en WebSphere Application Server v6.x, necesita DB2 Versión 8.1.6 o una versión superior. Si está ejecutando esas aplicaciones en el Entorno de prueba WebSphere v5.x, necesita DB2 Versión 8.1.3 o una versión superior.

### Informix

El mínimo controlador JDBC Informix mínimo aceptable es 2.21.JC6. Este nivel de controlador no se ajusta a JDBC 3.0 y por lo tanto no soporta la opción hold en la sentencia EGL **open**. El controlador Informix que se ajusta a JDBC 3.0 puede estar ya disponible y debe soportar la opción hold.

### Oracle

El controlador JDBC empaquetado con Oracle 10i es aceptable.

Las reglas siguientes se aplican a cualquier controlador JDBC utilizado con EGL:

- El controlador debe soportar JDBC 2.0 o una versión superior
- El valor `java.sql.ResultSet.CONCUR_UPDATABLE` debe permitirse en estos contextos:
  - Como segundo argumento de `java.sql.Connection.createStatement(int,int)`
  - Como tercer argumento de `java.sql.Connection.prepareStatement(String,int,int)` y `java.sql.Connection.prepareCall(String,int,int)`
- Si desea dar soporte a la opción hold en la sentencia EGL **open**, el controlador debe soportar JDBC 3.0 y el valor `java.sql.ResultSet.HOLD_CURSORS_OVER_COMMIT` debe permitirse en estos contextos:
  - Como tercer argumento de `java.sql.Connection.createStatement(int,int,int)`
  - Como cuarto argumento de `java.sql.Connection.prepareStatement(String,int,int,int)` y `java.sql.Connection.prepareCall(String,int,int,int)`

Para cualquier sistema de gestión de base de datos, los controladores JDBC de terceros son aceptables.

### Tareas relacionadas

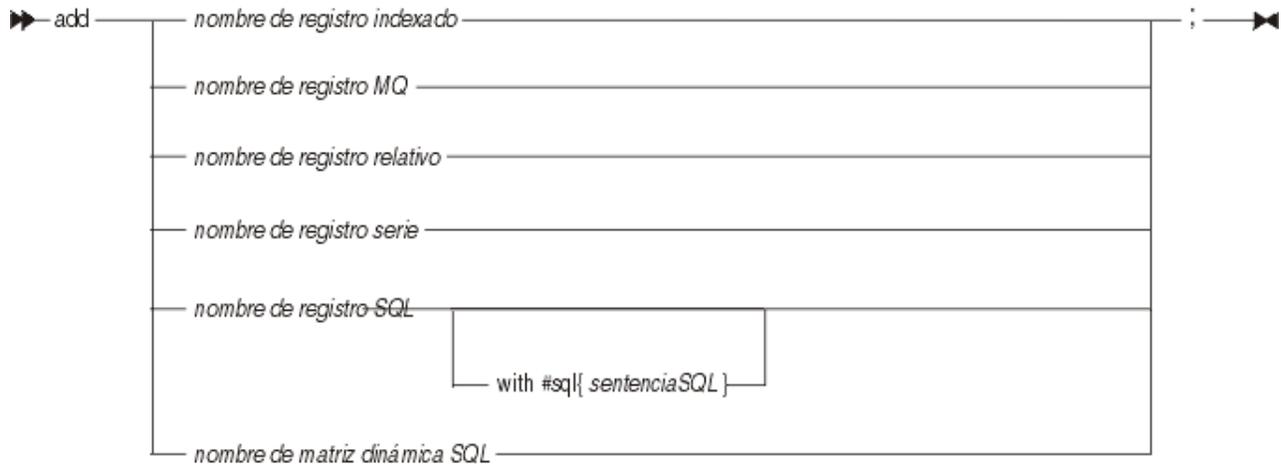
“Establecer una conexión JDBC J2EE” en la página 362

“Cómo se realiza una conexión JDBC estándar” en la página 263

## Palabra clave

### add

La sentencia EGL **add** coloca un registro en un archivo, una cola de mensajes o una base de datos; o coloca un conjunto de registros en una base de datos.



#### *nombre de registro*

Nombre del objeto de E/S que debe añadirse; un registro indexado, MQ, relativo, serie o SQL

#### **with #sql{ sentencia SQL }**

Una sentencia SQL INSERT explícita. No deje espacios después de #sql.

#### *nombre de matriz dinámica SQL*

El nombre de una matriz dinámica de registros SQL. Los elementos se insertan en la base de datos, cada uno en la posición especificada por los valores de clave específicos del elemento. La operación se detiene cuando se produce el primer error o cuando se insertan todos los elementos.

A continuación se ofrece un ejemplo:

```
if (userRequest == "A")
 try
 add record1;
 onException
 myErrorHandler(12);
end
end
```

El comportamiento de la sentencia **add** depende del tipo de registro. Para obtener detalles acerca del proceso SQL, consulte el tema *Registro SQL*.

### Registro indexado

Si añade un registro indexado, la clave del registro determina la posición lógica del registro en el archivo. La adición de un registro a una posición de archivo que ya se esté utilizando provoca un error grave de E/S UNIQUE o (si se permiten duplicaciones) el error leve de E/S DUPLICATE.

### Registro MQ

Al añadir un registro MQ, el registro se coloca al final de la cola. Esta colocación se produce debido a que la sentencia **add** invoca una o varias llamadas MQSeries:

- MQCONN conecta el código generado con el gestor de colas por omisión y se invoca cuando no hay ninguna conexión activa
- MQOPEN establece una conexión con la cola y se invoca cuando hay una conexión activa, pero la cola no está abierta
- MQPUT pone el registro en la cola y siempre se invoca a menos que se produzca un error en una llamada anterior de MQSeries

### Registro relativo

Si añade un registro relativo, el elemento de clave especifica la posición del registro en el archivo. Sin embargo, la adición de un registro a una posición de archivo que ya se esté utilizando provoca un error grave de E/S UNIQUE.

El elemento de clave de registro debe estar disponible para cualquier función que utilice el registro y puede ser cualquiera de los siguientes:

- Un elemento del mismo registro
- Un elemento de un registro que sea global con respecto al programa o local con respecto a la función que ejecuta la sentencia **add**
- Un elemento de datos que sea global con respecto al programa o local con respecto a la función que ejecuta la sentencia **add**

### Registro serie

Al añadir un registro serie, el registro se coloca al final del archivo.

Si el programa generado finaliza un registro serie y luego emite una sentencia **get next** para el mismo archivo, el programa cierra y vuelve a abrir el archivo antes de ejecutar la sentencia **get next**. Por tanto, una sentencia **get next** que sigue a una sentencia **add** lee el primer registro del archivo. Este comportamiento también se produce cuando las sentencias **get next** y **add** se encuentran en programas diferentes, y un programa llama al otro.

Es aconsejable evitar que el mismo archivo esté abierto en más de un programa simultáneamente.

### Registro SQL

A continuación se indican algunas condiciones de error:

- Se especifica una sentencia SQL de un tipo que no es INSERT
- Se especifica alguna, pero no todas las cláusulas de una sentencia SQL INSERT
- Se especifica una sentencia SQL INSERT (o se acepta una sentencia SQL implícita) que tiene alguna de estas características:
  - Está relacionada con más de una tabla SQL
  - Incluye sólo variables de lenguaje principal que ha declarado como de sólo lectura
  - Está asociada con una columna que no existe o que es incompatible con la variable de lenguaje principal relacionada

Al añadir un registro SQL sin especificar una sentencia SQL explícita, el resultado es el siguiente:

- El formato de la sentencia SQL INSERT generada es como este:

```
INSERT INTO nombre Tabla
 (columna01, ... columnaNN)
 values (:elementoRegistro01, ... :elementoRegistroNN)
```

- El valor de clave del registro determina la posición lógica de los datos en la tabla. Un registro que no tenga una clave se maneja de acuerdo con la definición de tabla SQL y las normas de la base de datos.
- Como resultado de la asociación de elementos de registro y columnas de tabla SQL en el componente de registro, el código generado coloca los datos de cada elemento de registro en la columna de tabla SQL relacionada.
- Si ha declarado un elemento de registro como de sólo lectura, la sentencia SQL INSERT generada no incluye ese elemento de registro, y el sistema de gestión de bases de datos establece el valor de la columna de tabla SQL relacionada en el valor por omisión especificado al definir la columna.

A continuación figura un ejemplo de utilización de una matriz dinámica de registros SQL:

```
try
 add employees;
onException
 sysLib.rollback();
end
```

#### **Conceptos relacionados**

“Referencias a componentes” en la página 21

“Tipos de registros y propiedades” en la página 135

“Soporte de SQL” en la página 229

#### **Tareas relacionadas**

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

#### **Consulta relacionada**

“close” en la página 568

“delete” en la página 571

“get” en la página 585

“get next” en la página 597

“get previous” en la página 602

“Manejo de excepciones” en la página 94

“execute” en la página 574

“Valores de error de E/S” en la página 536

“open” en la página 616

“prepare” en la página 630

“Sentencias EGL” en la página 88

“replace” en la página 632

“Propiedades de elementos SQL” en la página 67

## **call**

La sentencia EGL call transfiere el control a otro programa y, opcionalmente, pasa una serie de valores. El control vuelve al llamador cuando el programa llamado finaliza; si el programa llamado cambia los datos que se han pasado por medio de una variable, cambia también el área de almacenamiento disponible para el llamador.



### nombre de programa

Nombre del programa llamado. El programa es generado por EGL o se considera *definido externamente*.

El nombre especificado no puede ser una palabra reservada. Si el llamador debe llamar a un programa no EGL que tiene el mismo nombre que una palabra reservada EGL, utilice un nombre de programa diferente en la sentencia `call` y, a continuación, utilice un componente de opciones de enlace, elemento **callLink**, para especificar un alias, que es el nombre utilizado durante la ejecución.

Si el programa llamado es un programa Java, el nombre del programa llamado será sensible a las mayúsculas y minúsculas; *calledProgram* es distinto a *CALLEDPROGRAM*. En caso contrario, la determinación de si el nombre de programa es sensible a las mayúsculas y minúsculas depende del sistema en el que resida el programa llamado: sensible a las mayúsculas y minúsculas para UNIX, no sensible a las mayúsculas y minúsculas en otros casos.

En el depurador EGL, el nombre del programa llamado no es sensible a las mayúsculas y minúsculas.

### argumento

Una de una serie de referencias de valor, cada una de ellas separada del texto por una coma. Un argumento puede ser una variable primitiva, un formulario, un registro, un registro fijo, un literal no numérico, una constante no numérica o (si EGL tiene acceso al programa llamado durante la generación) una expresión de fecha y hora, numérica o de texto más compleja. No puede pasar un campo de tipo ANY, ArrayDictionary, Blob, Clob, DataTable o Dictionary. Tampoco puede pasar matrices de esos tipos o registros que incluyan cualquiera de esos tipos.

### externallyDefined

Indicador de que el programa está definido externamente. Este indicador sólo está disponible si establece la propiedad de proyecto para la compatibilidad con VisualAge Generator.

Es aconsejable que un programa no generado por EGL se identifique como definido externamente no en la sentencia `call`, sino en el componente de opciones de enlace utilizado durante la generación. (La propiedad relacionada se encuentra en el componente de opciones de enlace, elemento `callLink`, y también se denomina **externallyDefined**).

### noRefresh

Indicador de que debe evitarse una renovación de pantalla cuando el programa llamado devuelve el control.

El indicador está soportado (durante el desarrollo) si la propiedad de programa **VAGCompatibility** está seleccionada o (durante la generación) si la opción del descriptor de construcción **VAGCompatibility** está establecida en *yes*.

Este indicador es adecuado si el llamador está en una unidad de ejecución que presenta formularios de texto en una pantalla y se produce alguna de estas situaciones:

- El programa llamado no presenta un formulario de texto; o
- El llamador escribe un formulario de texto de pantalla completa después de la llamada.

Es aconsejable indicar la preferencia de la renovación de pantalla no en la sentencia **call**, sino en el componente de opciones de enlace utilizado durante la generación. (La propiedad relacionada se encuentra en el componente de opciones de enlace, elemento `callLink`, y se denomina **refreshScreen**).

A continuación se ofrece un ejemplo:

```
if (userRequest == "C")
 try
 call programA;
 onException
 myErrorHandler(12);
 end
end
```

El número, tipo y orden de los argumentos de una sentencia `call` debe corresponder al número, tipo y orden de los valores esperados por el programa llamado.

Es muy aconsejable que el número de bytes pasados en cada argumento sea el mismo que el número de bytes esperado. Una discrepancia de longitudes sólo provoca un error si la corrección de la discrepancia en tiempo de ejecución provoca una discrepancia de tipos.

- Si el programa Java llamado recibe demasiado pocos bytes, el final de los datos pasados se rellena con blancos.
- Si el programa Java llamado recibe demasiados bytes, el final de los datos pasados se trunca.

Un error si se añaden blancos a un elemento de datos de tipo `NUM`, por ejemplo, pero no si se añaden blancos a un elemento de datos de tipo `CHAR`.

Los literales y las constantes están sujetos a las siguientes normas:

- El tamaño de un literal o constante que se pasa debe ser igual al tamaño del parámetro receptor
- Un literal o constante numérica no puede pasarse como argumento
- Un literal o constante que incluya sólo caracteres de un solo byte puede pasarse a un parámetro de tipo `CHAR` o `MBCHAR`
- Un literal o constante que incluya sólo caracteres de doble byte puede pasarse sólo a un parámetro de tipo `DBCHAR`
- Un literal o constante que incluya una combinación de caracteres de un solo byte y de doble byte puede pasarse a un parámetro de tipo `MBCHAR`

Las llamadas recursivas están soportadas en todos los sistemas destino.

La llamada resulta afectada por el componente de opciones de enlace, si existe, utilizado durante la generación. (Puede incluir un componente de opciones de enlace estableciendo la opción **linkage** del descriptor de construcción).

Para obtener detalles acerca de la opción `linkage`, consulte el apartado *Componente de opciones de enlace*.

### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### Consulta relacionada

“Sentencias EGL” en la página 88

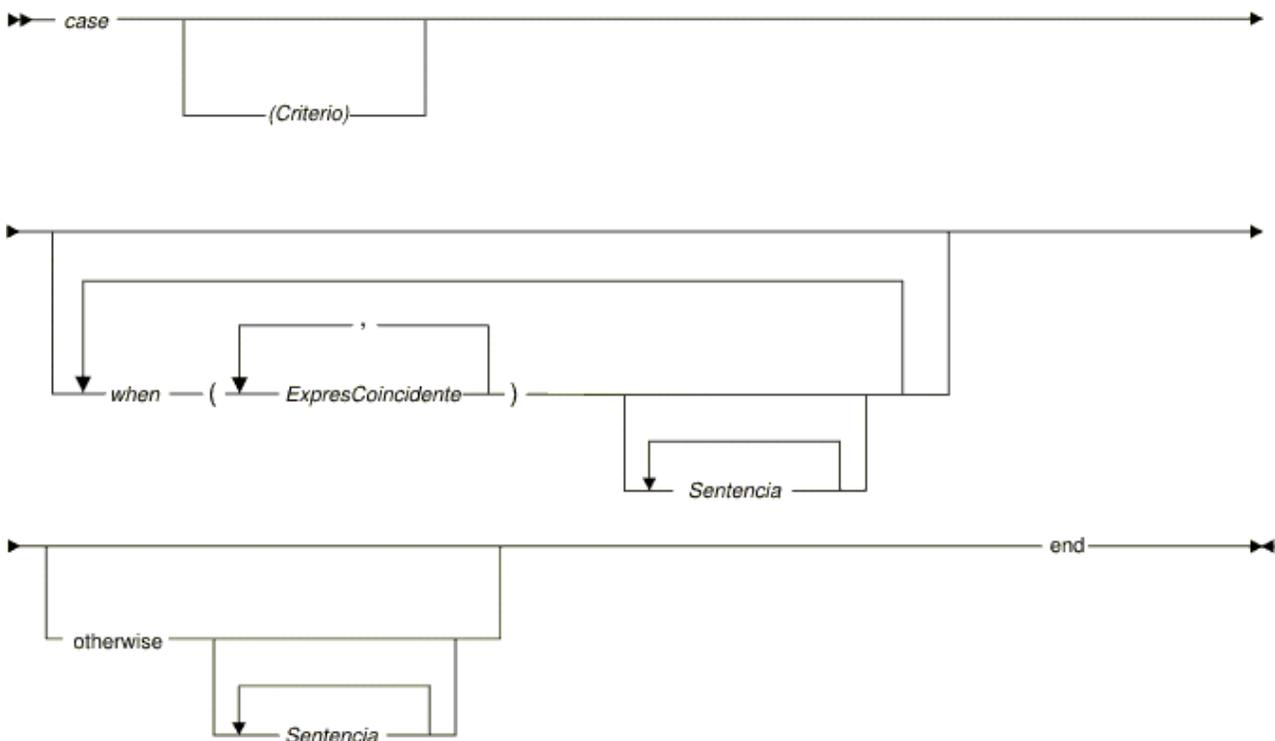
“Manejo de excepciones” en la página 94

“linkage” en la página 397

“Tipos primitivos” en la página 34

## case

La sentencia EGL **case** marca el inicio de varios conjuntos de sentencias, entre los que sólo se ejecuta uno de los conjuntos como máximo. La sentencia **case** es equivalente a una sentencia C o Java **switch** que tengan una interrupción al final de cada cláusula case.



### *criterio*

Elemento, constante, expresión, literal o variable de sistema, incluyendo ConverseVar.eventKey o sysVar.systemType.

Si especifica *criterio*, cada una de las cláusulas when subsiguientes (si las hay) debe contener una o varias instancias de la *expresiónCoincidente*. Si no especifica *criterio*, cada una de las cláusulas when subsiguientes (si las hay) debe contener una *expresión lógica*.

### **when**

El principio de una cláusula que sólo se invoca en estos casos:

- Ha especificado un *criterio* y la cláusula when es la primera que contiene una *expresiónCoincidente* que es igual al *criterio*; o
- No ha especificado un *criterio* y la cláusula when es la primera que contiene una *expresión lógica* que se evalúa en true.

Si desea que la cláusula `when` no tenga ninguna efecto cuando se invoca, codifíquela sin sentencias EGL.

Una sentencia **case** puede tener cualquier número de cláusulas `when`.

#### *expresiónCoincidente*

Uno de los siguientes valores:

- Una expresión numérica o de serie
- Un símbolo para la comparación con `ConverseVar.eventKey` o `sysVar.systemType`

El tipo primitivo del valor *expresiónCoincidente* debe ser compatible con el tipo primitivo del valor de criterio. Para obtener detalles acerca de la compatibilidad, consulte el apartado *Expresiones lógicas*.

#### *expresiónLógica*

Una *expresión lógica*.

#### *sentencia*

Una sentencia EGL.

#### **otherwise**

El principio de una cláusula que se invoca si no se ejecuta ninguna cláusula `when`.

Una vez que las sentencias han ejecutado una cláusula `when` u `otherwise`, el control pasa a la sentencia EGL que sigue inmediatamente al final de la sentencia **case**. El control no pasa a la siguiente cláusula `when` en ninguna circunstancia. Si no se ha invocado ninguna cláusula `when` y no se utiliza ninguna cláusula por omisión, el control también pasa a la sentencia inmediatamente posterior al final de la sentencia **case** y no se produce ninguna situación de error.

A continuación se ofrece un ejemplo de sentencia **case**:

```
case (myRecord.requestID)
 when (1)
 myFirstFunction();
 when (2, 3, 4)
 try
 call myProgram;
 onException
 myCallFunction(12);
 end
 otherwise
 myDefaultFunction();
end
```

Si una sola cláusula incluye varias instancias de *expresiónCoincidente* (2, 3, 4 en el ejemplo anterior), la evaluación de dichas instancias se realiza de izquierda a derecha, y se detiene en cuanto encuentra una *expresiónCoincidente* que corresponde al valor de criterio.

#### **Tareas relacionadas**

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

#### **Consulta relacionada**

“Sentencias EGL” en la página 88

“Expresiones lógicas” en la página 497

“eventKey” en la página 923

“systemType” en la página 937

## close

La sentencia EGL **close** desconecta una impresora; o cierra el archivo o cola de mensajes asociada con un registro dado; o, en el caso de un registro SQL, cierra el cursor que se abrió mediante una sentencia EGL **open** o **get**.



### *nombre*

Nombre del objeto de E/S asociado con el recurso que se cierra; dicho objeto es un formulario de impresión, un registro indexado, MQ, relativo, serie o SQL

### *identificadorConjuntoResultados*

Sólo para procesos SQL, un ID que conecta la sentencia **close** con una sentencia **get** o **open** ejecutada antes en el mismo programa. Para obtener detalles, consulte el apartado *resultSetID*.

### Ejemplo:

```
if (userRequest == "C")
 try
 close fileA;
 onException
 myErrorHandler(12);
end
end
```

El comportamiento de una sentencia **close** depende del tipo de objeto de E/S.

### **Registro indexado, serie o relativo**

Cuando el usuario utiliza el nombre de un registro indexado, serie o relativo en una sentencia **close**, EGL cierra el archivo asociado con ese registro.

Si hay un archivo abierto y se utiliza el elemento *fileAssociation* para cambiar el nombre de recurso asociado con ese archivo, EGL cierra el archivo automáticamente antes de ejecutar la sentencia siguiente que afecta al archivo. Para obtener detalles, consulte el apartado *resourceAssociation*.

EGL también cierra los archivos abiertos cuando finaliza el programa.

### **Registro MQ**

Cuando el usuario utiliza el nombre de un registro MQ en una sentencia **close**, EGL se asegura de que se ejecuta el mandato MQSeries MQCLOSE para la cola de mensajes asociada con ese registro.

## Formulario de impresión

Si el objeto de E/S es un formulario de impresión, la sentencia `close` emite una instrucción de alimentación de papel y se desconecta de la impresora o (si el formulario de impresión se almacena en spool en un archivo) cierra el archivo.

Antes de utilizar `ConverseVar.printerAssociation` para cambiar el destino de impresión, cierre la impresora o el archivo especificado por el valor actual de `ConverseVar.printerAssociation`. Emita una opción de sentencia `close` para cada destino de impresión, ya que puede haber varias impresoras o archivos de impresión abiertos simultáneamente.

El entorno de ejecución de EGL se asegura de que todas las impresoras estén cerradas cuando el programa finaliza.

## Registro SQL

Cuando el usuario utiliza el nombre de un registro SQL en una sentencia `close`, EGL cierra el cursor SQL que está abierto para ese registro.

EGL cierra automáticamente un cursor en estos casos:

- Un bucle de proceso de cursor sigue a una sentencia `open` y continúa hasta que una condición de Ningún registro encontrado (NRF) indica que se han procesado todas las filas del conjunto
- EGL ejecuta una sentencia `get` para un registro SQL cuando se lee una sola fila y no se han especificado `forUpdate` ni `singleRow` como opción
- EGL ejecuta una sentencia `replace` o `delete` que utiliza el cursor abierto por una sentencia `get`; en este caso, se ha especificado `forUpdate` como opción en la sentencia `get`
- EGL empieza a procesar una sentencia `open` o `get` para un registro que está asociado con un cursor abierto; el cierre precede a los demás procesos
- El programa ejecuta `sysLib.commit` o `sysLib.rollback`

EGL cierra todos los cursores abiertos en este caso:

- El programa es de tipo `textUI` y realiza un compromiso automático antes de invertir un formulario; para obtener detalles acerca de los programas `textUI` y de la sentencia `converse`, consulte el apartado *Segmentación*

## Conceptos relacionados

“Tipos de registros y propiedades” en la página 135

“`resultSetID`” en la página 743

“Segmentación en aplicaciones de texto” en la página 160

“Soporte de SQL” en la página 229

## Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

## Consulta relacionada

“`add`” en la página 561

“`delete`” en la página 571

“Sentencias EGL” en la página 88

“Manejo de excepciones” en la página 94

“`execute`” en la página 574

“`get`” en la página 585

“`get next`” en la página 597

“`get previous`” en la página 602

“Valores de error de E/S” en la página 536  
“open” en la página 616  
“prepare” en la página 630  
“replace” en la página 632  
“recordName.resourceAssociation” en la página 859  
“Propiedades de elementos SQL” en la página 67  
“commit()” en la página 893  
“rollback()” en la página 905  
“printerAssociation” en la página 923  
“terminalID” en la página 938

## continue

La sentencia EGL **continue** transfiere el control al final de una sentencia **for**, **forEach** o **while** que contiene la sentencia **continue**. La ejecución de la sentencia continúa o finaliza dependiendo de la prueba lógica realizada habitualmente al inicio de la sentencia contenedora.

La sentencia **continue** debe estar en la misma función que la sentencia contenedora.



### for, forEach o while

Identifica la sentencia contenedora más interior del tipo especificado. Si especifica uno de estos tipos de sentencia, la sentencia **continue** debe estar contenida en una sentencia de ese tipo. Si no especifica un tipo de sentencia, el resultado es el siguiente:

- La sentencia **continue** transfiere el control al final de la sentencia contenedora **for**, **forEach** o **while** más interior
- La sentencia **continue** debe estar contenida en una sentencia de uno de esos tipos.

### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### Consulta relacionada

“Sentencias EGL” en la página 88

## converse

La sentencia EGL **converse** presenta un formulario de texto en una aplicación de texto.

El programa espera una respuesta del usuario, recibe el formulario de texto del usuario y continúa el proceso con la sentencia que sigue a la sentencia **converse**.

Para obtener una visión general del proceso de formularios de texto, consulte estas páginas por orden:

1. Formularios de texto
2. Segmentación

►► `converse` `NomFormTexto` ; ►►

#### *nombreFormularioTexto*

Nombre del formulario de texto que es visible para el programa. Para obtener detalles acerca de la visibilidad, consulte el apartado *Referencias a componentes*.

A continuación se ofrece un ejemplo:

```
converse myTextForm;
```

Se aplican las siguientes normas:

- En relación a formularios de texto, una sentencia **converse** siempre es válida en un programa llamado; pero si se ejecuta un programa principal que está segmentado, la sentencia **converse** no es válida en estos tipos de código:
  - Una función que contiene parámetros, almacenamiento local o valores de retorno
  - Una función a la que invoca (directa o indirectamente) una función que contiene parámetros, almacenamiento local o valores de retorno

#### Conceptos relacionados

“Referencias a componentes” en la página 21

“Segmentación en aplicaciones de texto” en la página 160

## delete

La sentencia EGL **delete** elimina un registro de un archivo o una fila de una base de datos.

►► `delete` `Nombr reg indexado` `Nombre reg relativo` `Nombre reg SQL` `FromIdConjResult` ; ►►

#### *nombre de registro*

Nombre del objeto de E/S: un registro indexado, relativo o SQL asociado con el registro de archivo o la fila SQL que se suprime

*from IDconjuntoResultados*

ID que conecta la sentencia **delete** con una sentencia **get** o **open** ejecutada antes en el mismo programa. Para obtener detalles, consulte el apartado *resultSetID*.

A continuación se ofrece un ejemplo:

```
if (userRequest == "D")
 try
 get myRecord forUpdate;
 onException
 myErrorHandler(12); // sale del programa
 end

 try
 delete myRecord;
 onException
 myErrorHandler(16);
 end
end
```

El comportamiento de la sentencia **delete** depende del tipo de registro. Para obtener detalles acerca del proceso SQL, consulte el tema *Registro SQL*.

### Registro indexado o relativo

Si desea suprimir un registro indexado o relativo, haga lo siguiente:

- Emita una sentencia **get** para el registro y especifique la opción `forUpdate`
- Emita la sentencia **delete**, sin que intervenga ninguna operación de E/S en el mismo archivo

Después de emitir la sentencia **get**, el resultado de la próxima operación de E/S en el mismo archivo será el siguiente:

- Si la próxima operación de E/S es una sentencia **replace** en el mismo registro EGL, se cambia el registro del archivo
- Si la próxima operación de E/S es una sentencia **delete** en el mismo registro EGL, el registro del archivo se marca para supresión
- Si la próxima operación de E/S es una sentencia **get** en el mismo archivo (con la opción `forUpdate`), una sustitución o supresión subsiguiente es válida en el registro de archivo de lectura recién creado
- Si la próxima operación de E/S es una sentencia **get** en el mismo archivo (sin la opción `forUpdate`) o es un cierre en el mismo archivo, el registro de archivo se libera sin cambios.

Encontrará los detalles sobre la opción `forUpdate` en *get*.

### Registro SQL

En el caso de procesos SQL, debe utilizar la opción `forUpdate` en una sentencia EGL **get** o **open** para recuperar una fila para la supresión subsiguiente:

- Puede emitir una sentencia **get** para recuperar la fila; o
- Puede emitir una sentencia **open** para seleccionar un conjunto de filas y, a continuación, invocar una sentencia **get next** para recuperar la fila en cuestión.

En cualquier caso, la sentencia EGL **delete** está representada en el código generado por una sentencia SQL DELETE que hace referencia a la fila actual en un cursor. No puede modificar esa sentencia SQL, cuyo formato es el siguiente:

```
DELETE FROM nombreTabla
WHERE CURRENT OF cursor
```

Si desea escribir su propia sentencia SQL DELETE, utilice la sentencia EGL **execute**.

No puede utilizar una sola sentencia EGL **delete** para suprimir filas de varias tablas SQL.

#### Conceptos relacionados

"Tipos de registros y propiedades" en la página 135

"resultSetID" en la página 743

"Unidad de ejecución" en la página 742

"Soporte de SQL" en la página 229

#### Tareas relacionadas

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

#### Consulta relacionada

"add" en la página 561

"close" en la página 568

"Sentencias EGL" en la página 88

"Manejo de excepciones" en la página 94

"execute" en la página 574

"get" en la página 585

"get next" en la página 597

"get previous" en la página 602

"Valores de error de E/S" en la página 536

"prepare" en la página 630

"open" en la página 616

"replace" en la página 632

"Propiedades de elementos SQL" en la página 67

## display

La sentencia EGL **display** añade un formulario de texto a un almacenamiento intermedio de tiempo de ejecución, pero no presenta datos en la pantalla. Para obtener detalles acerca del comportamiento en tiempo de ejecución, consulte el apartado *Formularios de texto*.

**Nota:** Si está en modalidad de compatibilidad de VisualAge Generator, puede emitir una sentencia con el siguiente formato:

```
display formularioImpresión;
```

*formularioImpresión*

Nombre del formulario de impresión que es visible para el programa.

En ese caso, **display** es equivalente a print.

► display — *NomFormTexto* ————— ; ————— ►

#### nombreFormularioTexto

Nombre del formulario de texto que es visible para el programa. Para obtener detalles acerca de la visibilidad, consulte el apartado *Referencias a componentes*.

## Conceptos relacionados

“Referencias a componentes” en la página 21

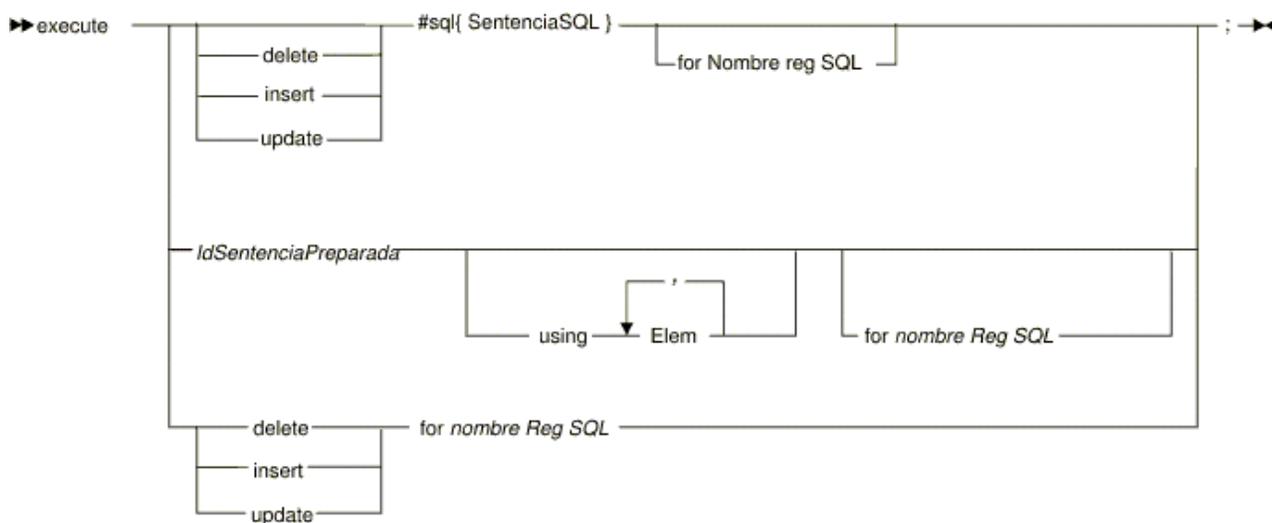
“Formularios de texto” en la página 158

## Consulta relacionada

“print” en la página 632

## execute

La sentencia EGL **execute** permite escribir una o varias sentencias SQL; en particular, sentencias de definición de datos SQL (de tipo CREATE TABLE, por ejemplo) y sentencias de manipulación de datos (de tipo INSERT o UPDATE, por ejemplo)



### **#sql{ sentenciaSQL }**

Una sentencia SQL explícita. Si desea que la sentencia SQL actualice o suprima una fila de un conjunto de resultados, codifique una sentencia SQL UPDATE o DELETE que incluya la siguiente cláusula:

```
WHERE CURRENT OF IDconjuntoResultados
```

### *IDconjuntoResultados*

El ID de conjunto de resultados especificado en la sentencia EGL open que ha hecho disponible el conjunto de resultados.

No deje espacios entre #sql y el corchete de apertura.

### **for nombre de registro SQL**

Nombre de un registro SQL.

Si especifica un tipo de sentencia (delete, insert o update), EGL utiliza el registro SQL para construir una sentencia SQL implícita, como se describe más adelante. En cualquier caso, puede utilizar el registro SQL para probar el resultado de la operación.

### *IDsentenciaPreparada*

Hace referencia a una sentencia EGL prepare que tiene el ID especificado. Si no hace referencia a una sentencia prepare, debe especificar una sentencia SQL explícita o una combinación de un registro SQL y un tipo de sentencia (delete, insert o update).

### **delete, insert, update**

Indica que EGL debe proporcionar una sentencia SQL implícita del tipo especificado. Si especifica un tipo de sentencia pero no un nombre de registro SQL, se produce un error de tiempo de declaración.

Si no establece un tipo de sentencia, debe especificar una sentencia SQL explícita o una referencia a una sentencia prepare.

Para obtener una visión general de las sentencias SQL implícitas, consulte el apartado *Soporte SQL*.

A continuación se ofrecen varios ejemplos de sentencias (suponiendo que employeeRecord esté en un registro SQL):

```
execute
 #sql{
 create table employee (
 empnum decimal(6,0) not null,
 empname char(40) not null,
 empphone char(10) not null)
 };

execute update for employeeRecord;

execute
 #sql{
 call aStoredProcedure(:argumentItem)
 };
```

Puede utilizar una sentencia **execute** para emitir sentencias SQL de los siguientes tipos:

- ALTER
- CALL
- CREATE ALIAS
- CREATE INDEX
- CREATE SYNONYM
- CREATE TABLE
- CREATE VIEW
- DECLARE tabla temporal global
- DELETE
- DROP INDEX
- DROP SYNONYM
- DROP TABLE
- DROP VIEW
- GRANT
- INSERT
- LOCK
- RENAME
- REVOKE
- SAVEPOINT
- SET
- SIGNAL
- UPDATE
- VALUES

No puede utilizar una sentencia **execute** para emitir sentencias SQL de los siguientes tipos:

- CLOSE
- COMMIT
- CONNECT

- CREATE FUNCTION
- CREATE PROCEDURE
- DECLARE CURSOR
- DESCRIBE
- DISCONNECT
- EXECUTE
- EXECUTE IMMEDIATE
- FETCH
- OPEN
- PREPARE
- ROLLBACK WORK
- SELECT
- INCLUDE SQLCA
- INCLUDE SQLDA
- WHENEVER

### **Sentencia SQL DELETE implícita**

El resultado de solicitar una sentencia SQL DELETE implícita es que una propiedad de registro SQL (**defaultSelectCondition**) determina qué filas de tabla se suprimen, siempre y cuando el valor de cada columna de clave de tabla SQL sea igual al valor del elemento de clave correspondiente del registro SQL. Si no especifica una clave de registro ni una condición de selección por omisión, se suprimen todas las filas de tabla.

La sentencia SQL DELETE implícita de un registro determinado es similar a la sentencia siguiente:

```
DELETE FROM nombreTabla
WHERE columnaClave01 = :elementoClave01
```

No puede utilizar una sola sentencia EGL para suprimir filas de más de una tabla de base de datos.

### **Sentencia SQL INSERT implícita**

Por omisión, el resultado de solicitar una sentencia SQL INSERT implícita es el siguiente:

- El valor de clave del registro determina la posición lógica de los datos en la tabla. Un registro que no tenga una clave se maneja de acuerdo con la definición de tabla SQL y las normas de la base de datos.
- Como resultado de la asociación de elementos de registro y columnas de tabla SQL en el componente de registro, el código generado coloca los datos de cada elemento de registro en la columna de tabla SQL relacionada.
- Si ha declarado un elemento de registro como de sólo lectura, la sentencia SQL INSERT generada no incluye ese elemento de registro, y el sistema de gestión de bases de datos establece el valor de la columna de tabla SQL relacionada en el valor por omisión especificado al definir la columna.

El formato de la sentencia SQL INSERT implícita es como este:

```
INSERT INTO nombreTabla
(columna01, ... columnaNN)
values (:elementoRegistro01, ... :elementoRegistroNN)
```

A continuación se indican algunas condiciones de error:

- Se especifica una sentencia SQL de un tipo que no es INSERT
- Se especifica alguna, pero no todas las cláusulas de una sentencia SQL INSERT

- Se especifica una sentencia SQL INSERT (o se acepta una sentencia SQL implícita) que tiene alguna de estas características:
  - Está relacionada con más de una tabla SQL
  - Incluye sólo variables de lenguaje principal que ha declarado como de sólo lectura
  - Está asociada con una columna que no existe o que es incompatible con la variable de lenguaje principal relacionada

### **Sentencia SQL UPDATE implícita**

Por omisión, el resultado de solicitar una sentencia SQL UPDATE implícita es el siguiente:

- Una propiedad de registro SQL (**defaultSelectCondition**) determina qué filas de tabla se seleccionan, siempre y cuando el valor de cada columna de clave de tabla SQL sea igual al valor del elemento de clave correspondiente del registro SQL. Si no especifica una clave de registro ni una condición de selección por omisión, se actualizan todas las filas de tabla.
- Como resultado de la asociación de elementos de registro y columnas de tabla SQL en la declaración de registro SQL, una columna de tabla SQL determinada recibe el contenido del elemento de registro relacionado. Sin embargo, si una columna de tabla SQL está asociada con un elemento de registro que es de sólo lectura, esa columna no se actualiza.

El formato de la sentencia SQL UPDATE implícita de un registro determinado es similar al de la sentencia siguiente:

```
UPDATE nombreTabla
SET columna01 = :elementoRegistro01,
 columna02 = :elementoRegistro01, ...
 columnaNN = :elementoRegistroNN
WHERE columnaClave01 = :elementoClave01
```

En cualquiera de los siguientes casos se produce un error:

- Todos los elementos están identificados como de sólo lectura
- La sentencia intenta actualizar más de una tabla SQL
- Un elemento cuyo valor se escribe en la base de datos está asociado con una columna que no existe en tiempo de ejecución o que es incompatible con ese elemento

### **Conceptos relacionados**

“Tipos de registros y propiedades” en la página 135

“Soporte de SQL” en la página 229

“Referencias a componentes” en la página 21

### **Tareas relacionadas**

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### **Consulta relacionada**

“add” en la página 561

“close” en la página 568

“delete” en la página 571

“Sentencias EGL” en la página 88

“Manejo de excepciones” en la página 94

“get” en la página 585

“get next” en la página 597

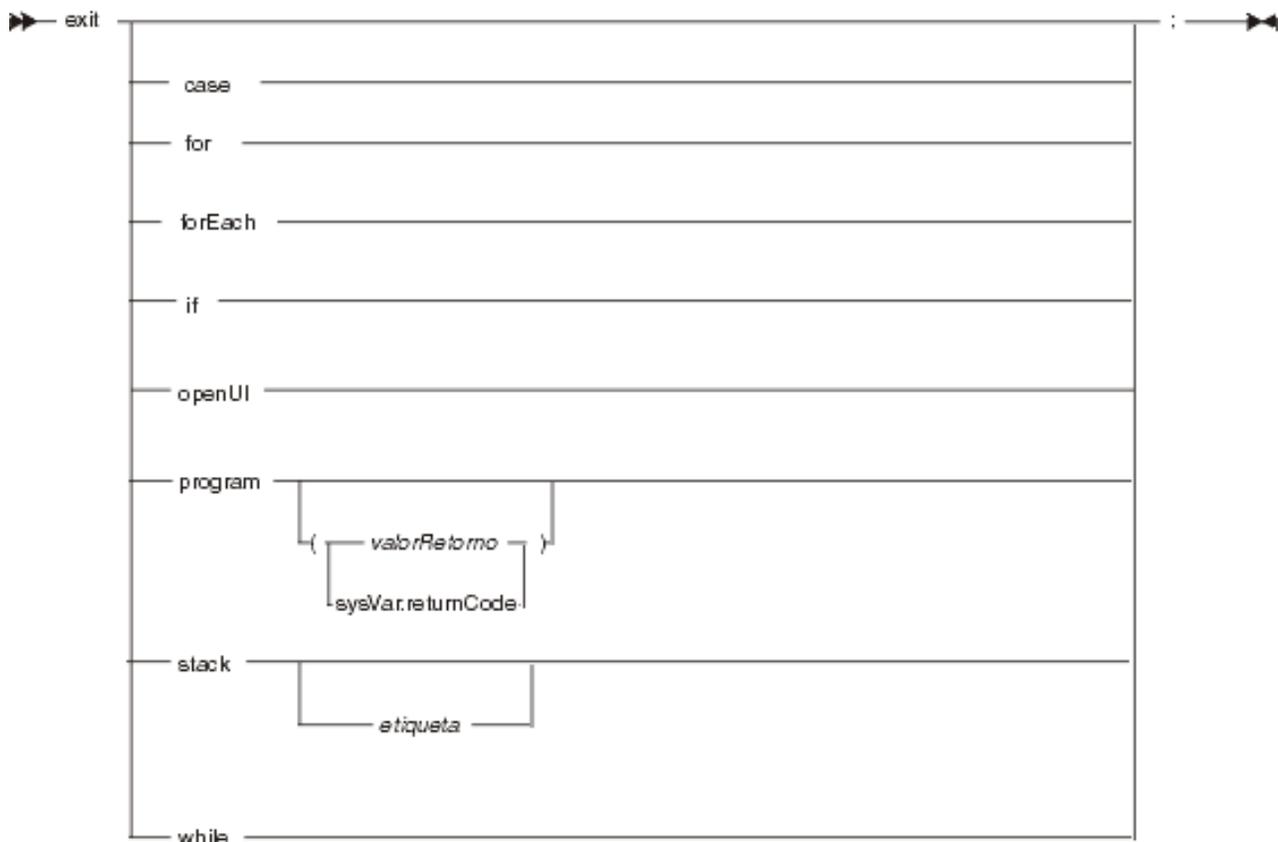
“get previous” en la página 602

“Valores de error de E/S” en la página 536

"open" en la página 616  
 "prepare" en la página 630  
 "replace" en la página 632  
 "Propiedades de elementos SQL" en la página 67  
 "terminalID" en la página 938

## exit

La sentencia **exit** de EGL sale del bloque especificado que, por omisión, es el bloque que contiene de inmediato la sentencia **exit**.



### case

Sale de la sentencia **case** especificada más recientemente en la que reside la sentencia **exit**. Continúa procesando después de la sentencia **case**.

Se produce un error si la sentencia **exit** no está dentro de una sentencia **case** que empiece en la misma función.

### for

Sale de la sentencia **for** especificada más recientemente en la que reside la sentencia **exit**. Continúa procesando después de la sentencia **for**.

Se produce un error si la sentencia **exit** no está dentro de una sentencia **for** que empiece en la misma función.

### forEach

Sale de la sentencia **forEach** especificada más recientemente en la que reside la sentencia **exit**. Continúa procesando después de la sentencia **forEach**.

Se produce un error si la sentencia **exit** no está dentro de una sentencia **forEach** que empiece en la misma función.

**if** Sale de la sentencia **if** especificada más recientemente en la que reside la sentencia **exit**. Continúa procesando después de la sentencia **if**.

Se produce un error si la sentencia **exit** no está dentro de una sentencia **if** que empiece en la misma función.

### **program**

Sale del programa.

El valor en la variable del sistema **sysVar.returnValue** se devuelve al sistema operativo en cualquiera de los siguientes casos:

- El programa finaliza con una sentencia **exit** que no incluye un código de retorno
- El programa finaliza con una sentencia **exit** que devuelve **sysVar.returnValue**
- El programa finaliza sin una sentencia **exit** de terminación

Si el programa finaliza con una sentencia **exit** de terminación que incluye un código de retorno que no sea **sysVar.returnValue**, se utiliza el valor especificado en lugar de cualquier valor que pueda haber en **sysVar.returnValue**.

### *valorRetorno*

Un entero literal o una expresión de elemento, constante, o numérica que se resuelve en un entero. El valor de retorno se pone a disposición del sistema operativo y debe estar en el rango de -2147483648 a 2147483647, ambos inclusive.

Para conocer más detalles sobre los valores de retorno, consulte la sección *sysVar.returnValue*.

### **sysVar.returnValue**

La variable del sistema que incluye el valor devuelto al sistema operativo.

Encontrará los detalles en *sysVar.returnValue*.

### **stack**

Devuelve el control a la función principal sin establecer un valor de retorno para la función actual.

Una sentencia con el formato *pila de salida* elimina todas las referencias a funciones intermedias en la *pila* del entorno de ejecución, que una lista de funciones; específicamente, la función actual más la serie de funciones cuya ejecución ha hecho posible la ejecución de la función actual.

La función principal puede haber invocado una función (ahora en la pila) y la invocación puede haber incluido un parámetro con el modificador **out** o **inOut**. En tales casos, la sentencia **exit** de la forma *exit stack* pone el valor de los parámetros a disposición de la función principal.

Si no especifica una *etiqueta* (como se describe más adelante), el proceso continúa en la sentencia posterior a la invocación de función ejecutada más recientemente en la función principal. Si especifica una etiqueta, el proceso continúa en la sentencia a continuación de la etiqueta en la función principal. La etiqueta puede ir delante o detrás de la invocación de función ejecutada más recientemente en la función principal.

Si especifica una sentencia de salida del formulario *pila de salida* en la función principal, se procesa la siguiente sentencia, incluso si especifica una etiqueta. Para obtener detalles sobre cómo ir a una etiqueta especificada en la función actual, consulte *goTo*.

#### *etiqueta*

Una serie de caracteres que se visualizan en la función principal y fuera de bloques, incluidos los siguientes:

- if
- else
- dentro de una sentencia **case**
- while
- try

Cuando se visualiza en la ubicación en la que continúa el proceso, la etiqueta va seguida de dos puntos. Encontrará los detalles sobre los caracteres válidos para la etiqueta en *Convenios de denominación*.

#### **Consulta relacionada**

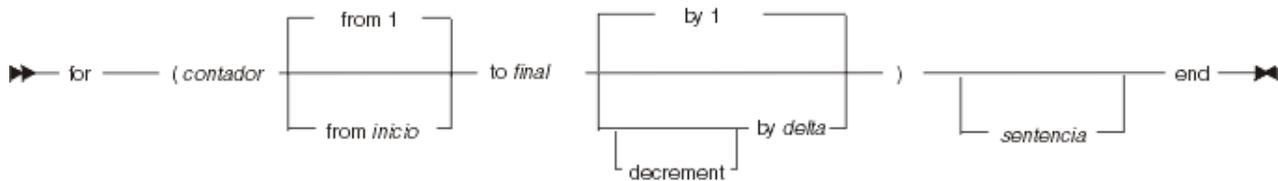
“goTo” en la página 608

“Convenios de denominación” en la página 672

“returnCode” en la página 933

## **for**

La palabra clave **for** inicia un bloque de sentencia que se ejecuta en un bucle tantas veces como una prueba de como resultado true. La prueba se realiza al principio del bucle e indica si el valor de un contador está dentro de un rango especificado. La palabra clave **end** marca el cierre de la sentencia **for**.



#### *contador*

Una variable numérica sin posiciones decimales. Las sentencias EGL en la sentencia **for** pueden cambiar el valor de *contador*.

#### **from inicio**

El valor inicial de *contador*. El valor inicial es 1 si no especifica una cláusula que empiece con **from**.

*inicio* puede ser cualquiera de los siguientes:

- Un literal entero
- Una variable numérica sin posiciones decimales
- Una expresión numérica que debe resolverse en un entero

#### **to final**

Si no especifica **decrement**, *final* es el límite superior de *contador* y si el valor de *contador* sobrepasa ese límite, la prueba mencionada anteriormente se resuelve como false, el bloque de sentencia no se ejecuta y la sentencia **for** finaliza.

Si especifica **decrement**, *final* es el límite inferior de *contador* y si el valor de *contador* está por debajo de ese límite, la prueba se resuelve como false, el bloque de sentencia no se ejecuta y la sentencia **for** finaliza.

*final* puede ser cualquiera de los siguientes:

- Un literal entero
- Una variable numérica sin posiciones decimales
- Una expresión numérica que debe resolverse en un entero

Las sentencias EGL en la sentencia **for** pueden cambiar el valor de *final*.

#### **by** *delta*

Si no especifica **decrement**, *delta* es el valor que se añade a *contador* después de ejecutar el bloque de sentencia EGL y antes de probar el valor de *contador*.

Si especifica **decrement**, *delta* es el valor que se resta de *contador* después de ejecutar el bloque de sentencia EGL y antes de probar el valor de *contador*.

*delta* puede ser cualquiera de estos:

- Un literal entero
- Una variable numérica sin posiciones decimales
- Una expresión numérica que debe resolverse en un entero

Las sentencias EGL en la sentencia **for** pueden cambiar el valor de *delta*.

#### *sentencia*

Una sentencia en el lenguaje EGL

A continuación se ofrece un ejemplo:

```
sum = 0;

// añade 10 valores a sum
for (i from 1 to 10 by 1)
 sum = inputArray[i] + sum;
end
```

#### **Tareas relacionadas**

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

#### **Consulta relacionada**

“Sentencias EGL” en la página 88

## **forEach**

La palabra clave EGL **forEach** marca el principio de un conjunto de sentencias que se ejecutan en un bucle. La primera iteración se produce solamente si un conjunto de resultados especificado está disponible. (Si el conjunto de resultados no está disponible, la sentencia falla con un error grave.) El bucle lee cada fila del conjunto de resultados hasta que tiene lugar uno de los sucesos siguientes:

- Se recuperan todas las filas
- Se ejecuta una sentencia **exit**
- Se produce un error grave o leve.



### *registroSQL*

Nombre de un registro SQL que se utiliza en una sentencia **open** ejecutada anteriormente. Debe especificar un registro SQL o un ID de conjunto de resultados y es recomendable que especifique el ID del conjunto de resultados.

### **from** *IDconjuntoResultados*

El identificador de conjunto de resultados que se utiliza en una sentencia **open** ejecutada anteriormente. Para obtener detalles, consulte el apartado *IDconjuntoResultados*.

### **into** ... *elemento*

Una cláusula INTO que identifica las variables de lenguaje principal EGL que reciben valores desde el cursor o procedimiento almacenado. En una cláusula como esta (que está fuera de un bloque **#sql{ }**), no incluya un punto y coma antes del nombre de una variable de lenguaje principal.

Una especificación de una cláusula INTO en este contexto altera temporalmente cualquier cláusula INTO identificada en la sentencia **open** relacionada.

### *sentencia*

Una sentencia en el lenguaje EGL

En la mayoría de los casos, el tiempo de ejecución de EGL emite una sentencia **close** implícita después de la última iteración de la sentencia **forEach**. Esa sentencia implícita modifica las variables del sistema SQL, razón por la que deberá guardar los valores de las variables del sistema relacionadas con SQL en el cuerpo de la sentencia **forEach**.

El tiempo de ejecución de EGL no emite una sentencia **close** implícita si la sentencia **forEach** finaliza debido a un error que no sea **noRecordFound**.

A continuación se proporciona un ejemplo que se trata en profundidad en *Ejemplos de SQL*:

```

VGVar.handleHardIOErrors = 1;

try
 open selectEmp
 with #sql{
 select empnum, empname
 from employee
 where empnum >= :empnum
 for update of empname
 }
 into empnum, empname;
onException
 myErrorHandler(6); // salir del programa
end

try
 forEach (from selectEmp)
 empname = empname + " " + "III";

 try
 execute
 #sql{

```

```

 update employee
 set empname = :empname
 where current of selectEmp
 };
onException
 myErrorHandler(10); // salir del programa
end
end // fin de forEach; el cursor se cierra automáticamente
// cuando se lee la última fila del conjunto de resultados

onException
 // el bloque de excepción relacionado con forEach no se ejecuta si la condición
 // es "sqlcode = 100", así que evite la prueba "if (sqlcode != 100)"
 myErrorHandler(8); // salir del programa
end

sysLib.commit();

```

### Conceptos relacionados

“resultSetID” en la página 743  
 “Soporte de SQL” en la página 229

### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### Consulta relacionada

“Sentencias EGL” en la página 88  
 “exit” en la página 578  
 “open” en la página 616  
 “Ejemplos de SQL” en la página 241

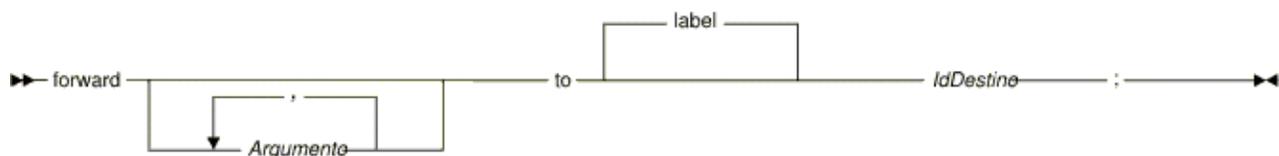
## forward

La sentencia EGL **forward** se invoca desde un PageHandler. La finalidad principal consiste en visualizar una página Web con información variable, pero la sentencia también puede invocar un servlet o un programa Java que se ejecute en el servidor de aplicaciones Web.

La sentencia actúa del siguiente modo:

1. Compromete recursos recuperables, cierra archivos y libera bloqueos
2. Reenvía en control
3. Finaliza el código que ejecuta la sentencia **forward**

El diagrama de sintaxis es el siguiente:



### argumento

Un elemento o registro que se pasa al código que se invoca. Los nombres de un argumento y su parámetro correspondiente deben ser los mismos en todos los casos. No pueden pasarse literales.

Si se invoca un PageHandler, los argumentos deben ser compatibles con los parámetros especificados para la función **onPageLoad** del PageHandler. La función (si existe) puede tener cualquier nombre válido y está referenciado en

la propiedad **OnPageLoadFunction** del PageHandler. Si se invoca un programa, los argumentos deben ser compatibles con los parámetros del programa.

Los siguientes detalles pueden ser de interés, en función de cómo utilice la tecnología:

- El argumento debe tener el mismo nombre que el parámetro correspondiente, ya que se utiliza como clave para almacenar y recuperar el valor del argumento en el servidor de aplicaciones Web.
  - En lugar de pasar el argumento, el invocante puede hacer lo siguiente antes de invocar la sentencia **forward**:
    - Coloca un valor en el bloque de petición invocando la función de sistema `J2EELib.setRequestAttr`; o
    - Coloca un valor en el bloque de sesión invocando la función de sistema `J2EELib.setSessionAttr`.
- En este caso, el receptor no recibe el valor como argumento, sino invocando la función de sistema adecuada:
- `J2EELib.getRequestAttr` (para acceder a los datos del bloque de petición); o
  - `J2EELib.getSessionAttr` (para acceder a los datos del bloque de sesión).
- Un elemento de carácter se pasa como objeto de tipo Java String.
  - Un registro se pasa como bean Java.

#### **to label** *IDdestino*

Especifica una etiqueta Java Server Faces (JSF), que identifica una correlación en un archivo de configuración basado en JSF de tiempo de ejecución. La correlación, a su vez, identifica el objeto que debe invocarse, ya sea un JSP (generalmente asociado con un PageHandler EGL), un programa EGL, un programa no EGL o un servlet. La palabra **label** es opcional e *IDdestino* es una serie entrecomillada.

#### **Consulta relacionada**

- “Invocaciones de función” en la página 518
- “`getRequestAttr()`” en la página 803
- “`getSessionAttr()`” en la página 803
- “`transferName`” en la página 939

## **freeSQL**

La sentencia **freeSQL** libera los recursos asociados a una sentencia SQL preparada dinámicamente, cerrando cualquier cursor abierto asociado con esa sentencia SQL.

```
►► freeSQL IDsentenciaPreparada _____ ; ◀◀
```

#### *IDsentenciaPreparada*

Un identificador que identifica una sentencia **prepare**. No se produce ningún error si la sentencia no se ha ejecutado anteriormente.

Después de emitir una sentencia **freeSQL** no puede ejecutar la sentencia **execute**, **open** ni **get** para la sentencia SQL preparada sin volver a emitir la sentencia **prepare**.

#### **Conceptos relacionados**

- “Soporte de SQL” en la página 229

### Consulta relacionada

“execute” en la página 574

“get”

“open” en la página 616

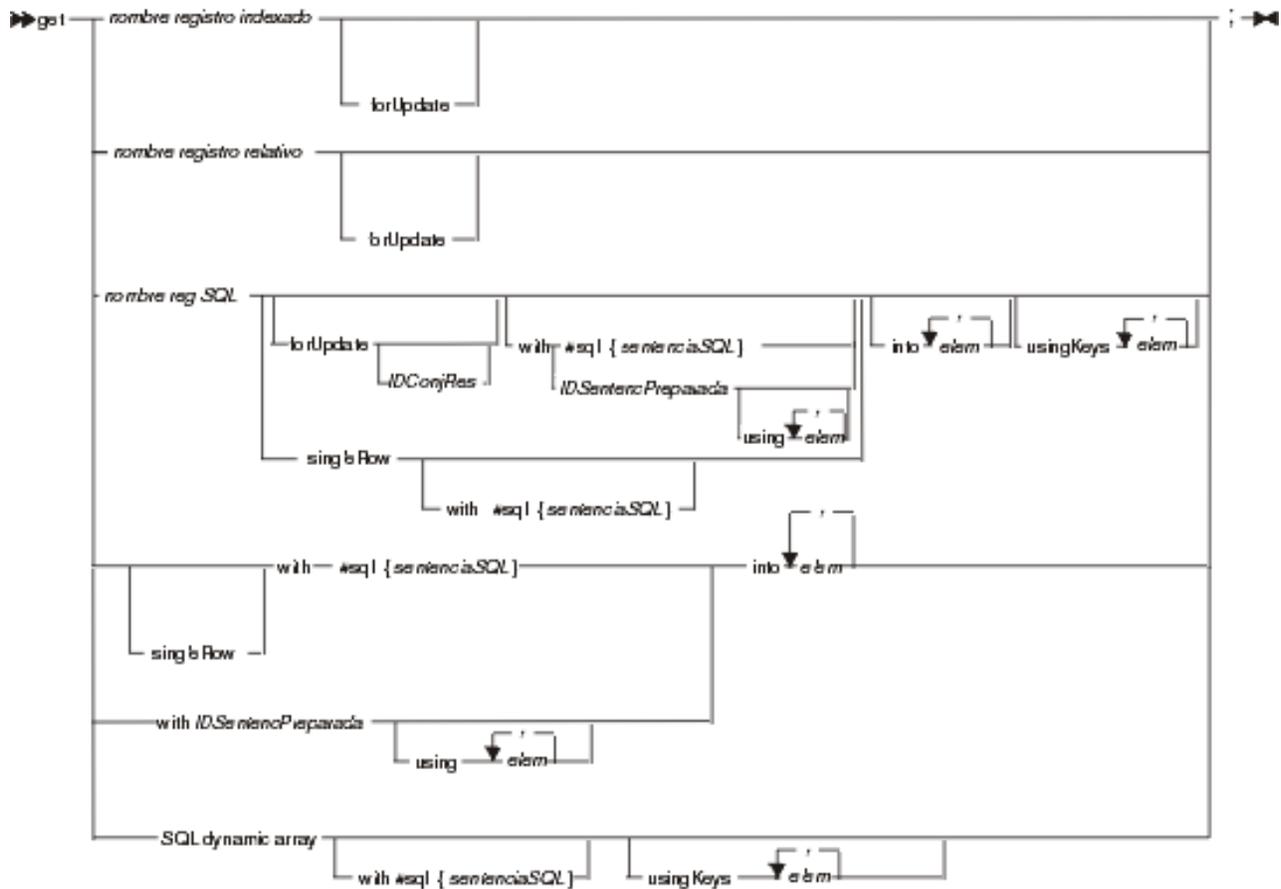
“prepare” en la página 630

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

## get

La sentencia EGL **get** recupera un solo registro de archivo o fila de base de datos y proporciona una opción que permite sustituir o suprimir los datos almacenados más tarde en el código. Además, esta sentencia permite recuperar un conjunto de filas de base de datos y sustituir cada fila sucesiva en el siguiente registro SQL de una matriz dinámica.

La sentencia **get** se identifica a veces como **get by key value** y es distinta de otras sentencias que empiezan por la palabra *get*.



### nombre de registro

Nombre de un objeto de E/S; un registro indexado, relativo o SQL. Para el proceso SQL, el nombre de registro es obligatorio si no se especifica la cláusula EGL INTO (descrita más adelante).

### forUpdate

Opción que permite utilizar una sentencia EGL posterior para sustituir o suprimir los datos recuperados del archivo o base de datos.

Si el recurso es recuperable (como en el caso de un archivo VSAM o una base de datos SQL), la opción **forUpdate** bloquea el registro de forma que otros programas no puedan cambiarlo hasta que se produzca un compromiso. Encontrará los detalles sobre el proceso de compromiso en *Unidad lógica de trabajo*.

#### *IDconjuntoResultados*

Un identificador de conjunto de resultados que se utiliza en una sentencia EGL **replace**, **delete** o **execute**, así como en una sentencia EGL **close**. Para obtener detalles, consulte el apartado *resultSetID*.

#### **singleRow**

Opción que provoca la generación de código SQL más eficiente, que es adecuada si está seguro de que el criterio de búsqueda de la sentencia **get** se aplica sólo a una fila y si no tiene intención de actualizar o suprimir la fila. Si especifica esta opción cuando el criterio de búsqueda se aplica a varias filas, se produce un error de E/S en tiempo de ejecución. Para obtener más detalles, consulte el apartado *Registro SQL*.

#### **#sql{ sentenciaSQL }**

Una sentencia SQL SELECT explícita, como se describe en el apartado *Soporte SQL*. No deje espacios entre **#sql** y el corchete de apertura.

#### **into ... elemento**

Una cláusula INTO que identifica las variables de lenguaje principal EGL que reciben valores desde una base de datos relacional. Esta cláusula es obligatoria al procesar SQL en cualquiera de estos casos:

- No se ha especificado un registro SQL; o
- Se especifican tanto un registro SQL como una sentencia SQL SELECT explícita, pero no hay una columna en la cláusula SQL SELECT asociada con un elemento de registro. (La asociación se encuentra en el componente de registro SQL, como se indica en el apartado *Propiedades de elementos SQL*).

En una cláusula como esta (que está fuera de un bloque **#sql{ }**), no incluya un punto y coma antes del nombre de una variable de lenguaje principal.

#### *IDsentenciaPreparada*

El identificador de una sentencia EGL **prepare** que prepara una sentencia SQL SELECT durante la ejecución. La sentencia **get** ejecuta la sentencia SQL SELECT dinámicamente. Para obtener detalles, consulte el apartado *prepare*.

#### **using ... elemento**

Una cláusula USING que identifica las variables de lenguaje principal EGL que quedan a disposición de la sentencia SQL SELECT preparada durante la ejecución. En una cláusula como esta (que está fuera de un bloque **sql-and-end**), no incluya un punto y coma antes del nombre de la variable de lenguaje principal.

#### **usingKeys ... elemento**

Identifica una lista de elementos de clave utilizados para construir el componente de clave-valor de la cláusula WHERE en la sentencia SQL implícita. La sentencia SQL implícita se utiliza durante la ejecución si no especifica una sentencia SQL explícita.

Si no especifica una cláusula **usingKeys**, el componente de clave-valor de la sentencia implícita se basa en el componente de registro SQL al que se hace referencia en la sentencia **get** o es la base de la matriz dinámica a la que se hace referencia en la sentencia **get**.

En el caso de una matriz dinámica, los elementos de la cláusula **usingKeys** (o las variables de lenguaje principal del registro SQL) *no* deben estar en el registro SQL que es la base de la matriz dinámica.

La información de **usingKeys** se pasa por alto si especifica una sentencia SQL explícita.

#### *matriz dinámica SQL*

El nombre de una matriz dinámica compuesta de registros SQL.

El ejemplo siguiente muestra cómo leer y sustituir un registro de archivo:

```
emp.empnum = 1; // establece la clave en el registro emp

try
 get emp forUpdate;
onException
 myErrorHandler(8); // sale del programa
end

emp.empname = emp.empname + " Smith";

try
 replace emp;
onException
 myErrorHandler(12);
end
```

La siguiente sentencia **get** utiliza el registro SQL emp al recuperar una fila de la base de datos, sin que sea posible la actualización o supresión subsiguiente:

```
try
 get emp singleRow into empname with
 #sql{
 select empname
 from Employee
 where empnum = :empnum
 };
onException
 myErrorHandler(8);
end
```

El ejemplo siguiente utiliza el mismo registro SQL para sustituir una fila SQL:

```
try
 get emp forUpdate into empname with
 #sql{
 select empname
 from Employee
 where empnum = :empnum
 };

onException
 myErrorHandler(8); // sale del programa
end

emp.empname = emp.empname + " Smith";

try
 replace emp;
onException
 myErrorHandler(12);
end
```

Los detalles de la sentencia **get** dependen del tipo de registro. Para obtener detalles acerca del proceso SQL, consulte el tema *Registro SQL*.

## Registro indexado

Si emite una sentencia **get** en un registro indexado, el valor de clave del registro determina qué registro se recupera del archivo.

Si desea sustituir o suprimir un registro indexado (o relativo), debe emitir una sentencia **get** para el registro y, a continuación, emitir la sentencia de cambio de archivo (**replace** o **delete**) sin que intervenga ninguna operación de E/S en el mismo archivo. Después de emitir la sentencia **get**, el resultado de la próxima operación de E/S en el mismo archivo será el siguiente:

- Si la próxima operación de E/S es una sentencia **replace** en el mismo registro EGL, se cambia el registro del archivo
- Si la próxima operación de E/S es una sentencia **delete** en el mismo registro EGL, el registro del archivo se marca para supresión
- Si la próxima operación de E/S es una sentencia **get** en un registro del mismo archivo e incluye la opción **forUpdate**, una sentencia **replace** o **delete** subsiguiente es válida en el registro de archivo de lectura recién creado
- Si la próxima operación de E/S es una sentencia **get** en el mismo registro EGL (sin la opción **forUpdate**) o es una sentencia **close** en el mismo archivo, el registro de archivo se libera sin cambios.

Si el archivo es un archivo VSAM, la sentencia de EGL **get** (con la opción **forUpdate**) impide que otros programas cambien el registro.

## Registro relativo

Si emite una sentencia **get** en un registro relativo, el elemento de clave asociado con el registro determina qué registro se recupera del archivo. El elemento de clave debe estar disponible para cualquier función que utilice el registro y puede ser cualquiera de los siguientes:

- Un elemento del mismo registro
- Un elemento de un registro que sea global con respecto al programa o local con respecto a la función que ejecuta la sentencia **get**
- Un elemento de datos que sea global con respecto al programa o local con respecto a la función que ejecuta la sentencia **get**

Si desea sustituir o suprimir un registro indexado (o relativo), debe emitir una sentencia **get** para el registro y, a continuación, emitir la sentencia de cambio de archivo (**replace** o **delete**) sin que intervenga ninguna operación de E/S en el mismo archivo. Después de emitir la sentencia **get**, el resultado de la próxima operación de E/S en el mismo archivo será el siguiente:

- Si la próxima operación de E/S es una sentencia **replace** en el mismo registro EGL, se cambia el registro del archivo
- Si la próxima operación de E/S es una sentencia **delete** en el mismo registro EGL, el registro del archivo se marca para supresión
- Si la próxima operación de E/S es una sentencia **get** en el mismo archivo (con la opción **forUpdate**), una sustitución o supresión subsiguiente es válida en el registro de archivo de lectura recién creado
- Si la próxima operación de E/S es una sentencia **get** en el mismo archivo (sin la opción **forUpdate**) o es un cierre en el mismo archivo, el registro de archivo se libera sin cambios.

## Registro SQL

La sentencia EGL **get** da como resultado una sentencia SQL **SELECT** en el código generado. Si especifica la opción **singleRow**, la sentencia SQL **SELECT** es una

sentencia autónoma. La sentencia SQL SELECT también puede ser una cláusula de un cursor, como se describe en el apartado *Soporte SQL*.

**Condiciones de error:** Las condiciones siguientes se encuentran entre las que no son válidas al utilizar una sentencia **get** para leer datos de una base de datos relacional:

- Se especifica una sentencia SQL de un tipo que no es SELECT
- Especifica una cláusula SQL INTO directamente en una sentencia SQL SELECT
- Además de una cláusula SQL INTO, especifica algunas cláusulas de una sentencia SQL SELECT, pero no todas
- Especifica (o acepta) una sentencia SQL SELECT que está asociada con una columna que no existe o que es incompatible con la variable de lenguaje principal relacionada

Las siguientes condiciones de error se encuentran entre las que se producen al utilizar la opción forUpdate:

- Especifica (o acepta) una sentencia SQL que muestra un intento de actualizar varias tablas; o
- Utiliza un registro SQL como objeto de E/S, y todos los elementos de registro son de sólo lectura.

La situación siguiente también provoca un error:

- Personaliza una sentencia EGL **get** con la opción forUpdate, pero no puede indicar que una columna de tabla SQL determinada está disponible para actualización; y
- La sentencia replace que está relacionada con la sentencia **get** intenta revisar la columna.

Puede resolver la discrepancia anterior de cualquiera de estas formas:

- Al personalizar la sentencia EGL **get**, incluya el nombre de columna en la sentencia SQL SELECT, cláusula FOR UPDATE OF; o
- Al personalizar la sentencia EGL replace, elimine la referencia a la columna en la sentencia SQL UPDATE, cláusula SET; o
- Acepte los valores por omisión para las sentencias **get** y **replace**.

**Sentencia SQL SELECT implícita:** Al especificar un registro SQL como objeto de E/S para la sentencia **get**, pero no especifica una sentencia SQL explícita, la sentencia SQL SELECT implícita tiene las siguientes características:

- La propiedad específica de registro denominada **defaultSelectCondition** determina qué fila de tabla se selecciona, siempre y cuando el valor de cada columna de clave de tabla SQL sea igual al valor del elemento de clave correspondiente del registro SQL. Si no especifica una clave de registro ni una condición de selección por omisión, se seleccionan todas las filas de tabla. Si por alguna razón se seleccionan varias filas de tabla, la primera fila recuperada se coloca en el registro.
- Como resultado de la asociación de elementos de registro y columnas de tabla SQL en la definición de registro, un elemento determinado recibe el contenido de la columna de tabla SQL relacionada.
- Si especifica la opción forUpdate, la sentencia SQL SELECT FOR UPDATE no incluye elementos de registro que sean de sólo lectura.

- La sentencia SQL SELECT de un registro determinado es similar a la sentencia siguiente, excepto que la cláusula SQL UPDATE sólo está presente si la sentencia **get** incluye la opción forUpdate:

```
SELECT columna01,
 columna02, ...
 columnaNN
FROM nombreTabla
WHERE columnaClave01 = :elementoClave01
FOR UPDATE OF
 columna01,
 columna02, ...
 columnaNN
```

La cláusula SQL INTO de la sentencia SQL SELECT autónoma o de la sentencia FETCH relacionada con el cursor es similar a esta:

```
INTO :elementoRegistro01,
 :elementoRegistro02, ...
 :elementoRegistroNN
```

EGL deriva la cláusula SQL INTO si el registro SQL va acompañado de una sentencia SQL SELECT explícita cuando no se ha especificado una cláusula INTO. Los elementos de la cláusula INTO derivada son aquellos que están asociados con las columnas listadas en la cláusula SELECT de la sentencia SQL. (La asociación de elementos y columnas se encuentra en el componente de registro SQL, como se indica en el apartado *Propiedades de elementos SQL*). es necesaria una cláusula EGL INTO si no hay una columna asociada con un elemento.

Si especifica una matriz dinámica de registros SQL como objeto de E/S para la sentencia **get**, pero no especifica una sentencia SQL explícita, la sentencia SQL SELECT implícita es similar a la descrita para un solo registro SQL, con estas diferencias:

- El componente de clave-valor de la consulta es un conjunto de relaciones que se basa en una condición de mayor-que-o-igual-que:
 

```
columnaClave01 >= :elementoClave01 &
columnaClave02 >= :elementoClave02 &
.
.
.
columnaClaveN >= :elementoClaveN
```
- Los elementos de la cláusula **usingKeys** (o las variables de lenguaje principal del registro SQL) *no* deben estar en el registro SQL que es la base de la matriz dinámica.

#### Conceptos relacionados

“Unidad lógica de trabajo” en la página 307  
 “Tipos de registros y propiedades” en la página 135  
 “Referencias a componentes” en la página 21  
 “resultSetID” en la página 743  
 “Soporte de SQL” en la página 229

#### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

#### Consulta relacionada

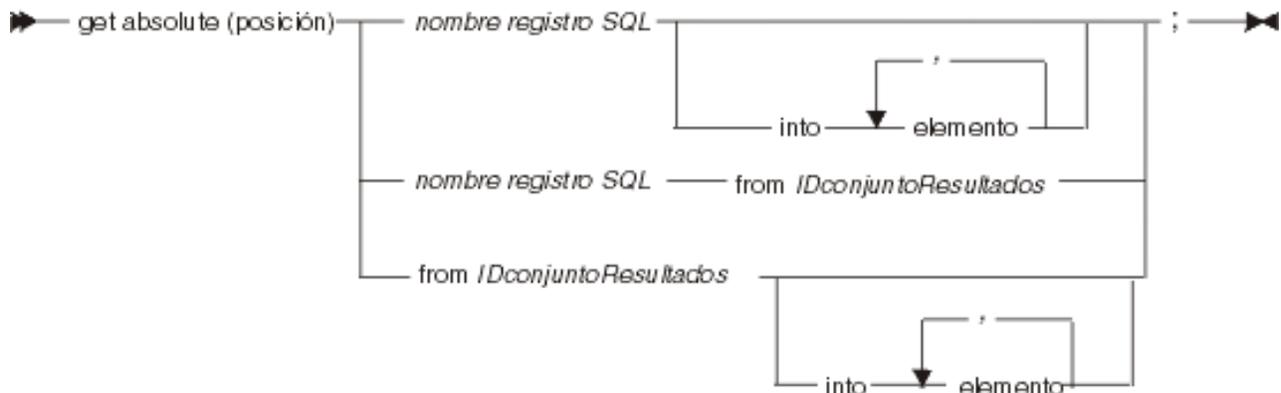
“add” en la página 561  
 “close” en la página 568  
 “delete” en la página 571

"Sentencias EGL" en la página 88  
 "Manejo de excepciones" en la página 94  
 "execute" en la página 574  
 "get next" en la página 597  
 "get previous" en la página 602  
 "Valores de error de E/S" en la página 536  
 "open" en la página 616  
 "prepare" en la página 630  
 "replace" en la página 632  
 "Propiedades de elementos SQL" en la página 67  
 "terminalID" en la página 938

## get absolute

La sentencia EGL **get absolute** lee una fila especificada numéricamente en un conjunto de resultados de base de datos relacional. La fila se identifica en relación con el principio del conjunto de resultados (si especifica un valor positivo) o con el final del conjunto de resultados (si especifica un valor negativo).

Solo puede utilizar esta sentencia si especificó la opción scroll en la sentencia **open** relacionada.



### *posición*

Literal o elemento entero.

Si el valor de *posición* es positivo, la fila se identifica en relación con el inicio del conjunto de resultados. Al especificar **get absolute 1**, por ejemplo, se recupera la primera fila y es equivalente a especificar **get first**. Si se especifica **get absolute 2** se recupera la segunda fila.

Si el valor de *posición* es negativo, la fila se identifica en relación con el final del conjunto de resultados. Al especificar **get absolute -1**, por ejemplo, se recupera la última fila y es equivalente a especificar **get last**. Si se especifica **get absolute -2** recupera de la segunda a la última fila.

Un valor cero para *posición* origina un error grave tal como se describe en la sección *Manejo de excepciones*.

### *nombre de registro*

Nombre de un registro SQL.

### **from** *IDconjuntoResultados*

Un ID que conecta la sentencia **get absolute** con una sentencia **open** ejecutada anteriormente en el mismo programa. Para obtener detalles, consulte el apartado *resultSetID*.

### **into**

Inicia una cláusula EGL *into*, que lista los elementos que recibe valores de una tabla de base de datos relacional.

### *elemento*

Un elemento que recibe el valor de una columna determinada. *No* especifique un signo de dos puntos (:) ante el nombre del elemento.

Si emite una sentencia **get absolute** para recuperar una fila seleccionada mediante una sentencia **open** que tiene la opción *forUpdate*, puede realizar cualquiera de estas acciones:

- Cambiar la fila con una sentencia EGL **replace**
- Eliminar la fila con una sentencia EGL **delete**
- Cambiar o eliminar la fila con una sentencia EGL **execute**

Una sentencia SQL *FETCH* representa la sentencia EGL **get absolute** en el código generado. El formato de la sentencia SQL generada no puede cambiarse, excepto para establecer la cláusula *INTO*.

Si emite una sentencia **get absolute** que intente acceder a una fila que no esté en el conjunto de resultados, el tiempo de ejecución EGL actúa de la forma siguiente:

- No copia datos del conjunto de resultados
- Deja el cursor abierto, con la posición del cursor inalterada
- Establece el registro SQL (si lo hay) en **noRecordFound**

Por lo general, si se produce un error y el proceso continúa, el cursor permanece abierto, con la posición del cursor inalterada.

Finalmente, al especificar *SQL COMMIT* o *sysLib.commit*, el código conserva la posición en el cursor declarado en la sentencia **open**, pero sólo en caso de que utilice la opción *hold* en la sentencia **open**.

### **Conceptos relacionados**

"resultSetID" en la página 743

"Soporte de SQL" en la página 229

### **Tareas relacionadas**

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

### **Consulta relacionada**

"delete" en la página 571

"Manejo de excepciones" en la página 94

"execute" en la página 574

"get" en la página 585

"get current" en la página 593

"get first" en la página 594

"get last" en la página 595

"get next" en la página 597

"get previous" en la página 602

"get relative" en la página 606

“Sentencias EGL” en la página 88

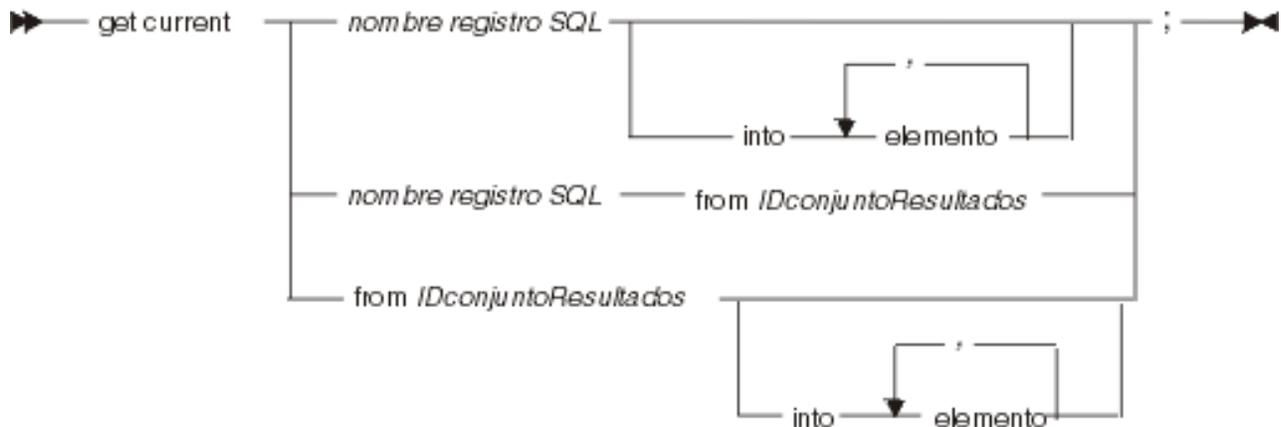
“open” en la página 616

“replace” en la página 632

## get current

La sentencia **get current** de EGL lee la fila en la que ya está situado el cursor en un conjunto de resultados de base de datos relacional.

Solo puede utilizar esta sentencia si especificó la opción scroll en la sentencia **open** relacionada.



*nombre de registro*

Nombre de un registro SQL.

**from** *IDconjuntoResultados*

Un ID que conecta la sentencia **get current** con una sentencia **open** ejecutada anteriormente en el mismo programa. Para obtener detalles, consulte el apartado *resultSetID*.

**into**

Inicia una cláusula EGL `into`, que lista los elementos que recibe valores de una tabla de base de datos relacional.

*elemento*

Un elemento que recibe el valor de una columna determinada. *No* especifique un signo de dos puntos (:) ante el nombre del elemento.

Si emite una sentencia **get current** para recuperar una fila seleccionada mediante una sentencia **open** que tiene la opción `forUpdate`, puede realizar cualquiera de estas acciones:

- Cambiar la fila con una sentencia EGL **replace**
- Eliminar la fila con una sentencia EGL **delete**
- Cambiar o eliminar la fila con una sentencia EGL **execute**

Una sentencia SQL `FETCH` representa la sentencia EGL **get current** en el código generado. El formato de la sentencia SQL generada no puede cambiarse, excepto para establecer la cláusula `INTO`.

Si se produce un error y el proceso continúa, el cursor permanece abierto.

Finalmente, al especificar SQL COMMIT o sysLib.commit, el código conserva la posición en el cursor declarado en la sentencia **open**, pero sólo en caso de que utilice la opción hold en la sentencia **open**.

#### Conceptos relacionados

"resultSetID" en la página 743

"Soporte de SQL" en la página 229

#### Tareas relacionadas

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

#### Consulta relacionada

"delete" en la página 571

"execute" en la página 574

"get" en la página 585

"get absolute" en la página 591

"get first"

"get last" en la página 595

"get next" en la página 597

"get previous" en la página 602

"get relative" en la página 606

"Sentencias EGL" en la página 88

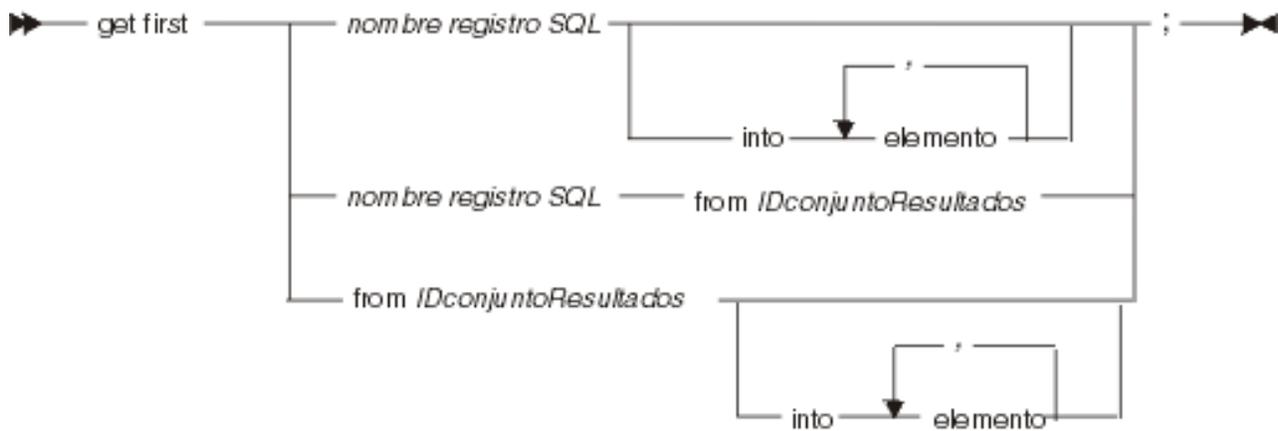
"open" en la página 616

"replace" en la página 632

## get first

La sentencia EGL **get first** lee la primera fila de un conjunto de resultados de base de datos relacional.

Solo puede utilizar esta sentencia si especificó la opción scroll en la sentencia **open** relacionada.



*nombre de registro*

Nombre de un registro SQL.

### **from** *IDconjuntoResultados*

Un ID que conecta la sentencia **get first** con una sentencia **open** ejecutada anteriormente en el mismo programa. Para obtener detalles, consulte el apartado *resultSetID*.

### **into**

Inicia una cláusula EGL **into**, que lista los elementos que recibe valores de una tabla de base de datos relacional.

### *elemento*

Un elemento que recibe el valor de una columna determinada. *No* especifique un signo de dos puntos (:) ante el nombre del elemento.

Si emite una sentencia **get first** para recuperar una fila seleccionada mediante una sentencia **open** que tiene la opción **forUpdate**, puede realizar cualquiera de estas acciones:

- Cambiar la fila con una sentencia EGL **replace**
- Eliminar la fila con una sentencia EGL **delete**
- Cambiar o eliminar la fila con una sentencia EGL **execute**

Una sentencia SQL **FETCH** representa la sentencia EGL **get first** en el código generado. El formato de la sentencia SQL generada no puede cambiarse, excepto para establecer la cláusula **INTO**.

Si se produce un error y el proceso continúa, el cursor permanece abierto.

Finalmente, al especificar **SQL COMMIT** o **sysLib.commit**, el código conserva la posición en el cursor declarado en la sentencia **open**, pero sólo en caso de que utilice la opción **hold** en la sentencia **open**.

### **Conceptos relacionados**

"resultSetID" en la página 743

"Soporte de SQL" en la página 229

### **Tareas relacionadas**

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

### **Consulta relacionada**

"delete" en la página 571

"execute" en la página 574

"get" en la página 585

"get absolute" en la página 591

"get current" en la página 593

"get last"

"get next" en la página 597

"get previous" en la página 602

"get relative" en la página 606

"Sentencias EGL" en la página 88

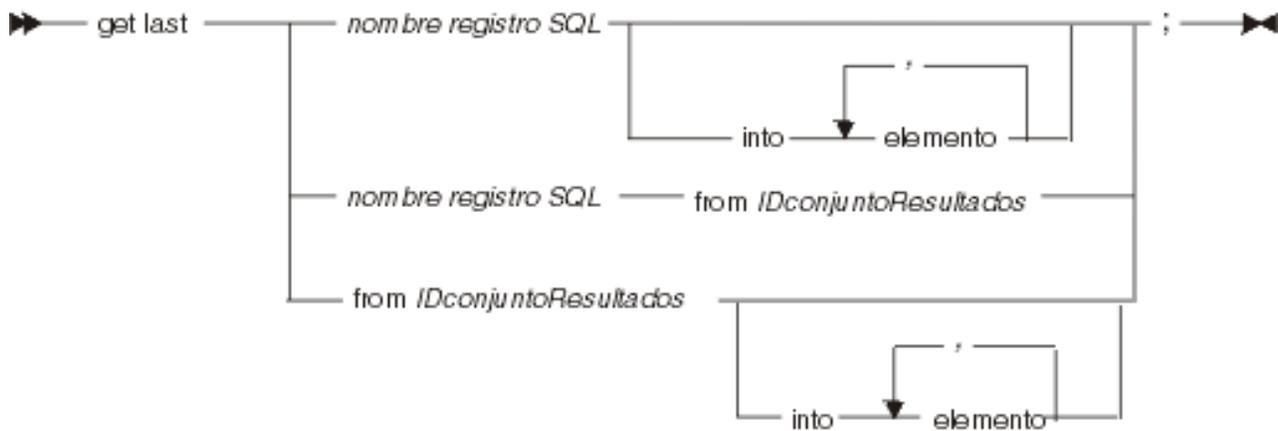
"open" en la página 616

"replace" en la página 632

## **get last**

La sentencia EGL **get last** lee la última fila de un conjunto de resultados de base de datos relacional.

Solo puede utilizar esta sentencia si especificó la opción scroll en la sentencia **open** relacionada.



*nombre de registro*

Nombre de un registro SQL.

**from** *IDconjuntoResultados*

Un ID que conecta la sentencia **get last** con una sentencia **open** ejecutada anteriormente en el mismo programa. Para obtener detalles, consulte el apartado *resultSetID*.

**into**

Inicia una cláusula EGL `into`, que lista los elementos que recibe valores de una tabla de base de datos relacional.

*elemento*

Un elemento que recibe el valor de una columna determinada. *No* especifique un signo de dos puntos (:) ante el nombre del elemento.

Si emite una sentencia **get last** para recuperar una fila seleccionada mediante una sentencia **open** que tiene la opción `forUpdate`, puede realizar cualquiera de estas acciones:

- Cambiar la fila con una sentencia EGL **replace**
- Eliminar la fila con una sentencia EGL **delete**
- Cambiar o eliminar la fila con una sentencia EGL **execute**

Una sentencia SQL `FETCH` representa la sentencia EGL **get last** en el código generado. El formato de la sentencia SQL generada no puede cambiarse, excepto para establecer la cláusula `INTO`.

Si se produce un error y el proceso continúa, el cursor permanece abierto.

Finalmente, al especificar `SQL COMMIT` o `sysLib.commit`, el código conserva la posición en el cursor declarado en la sentencia **open**, pero sólo en caso de que utilice la opción `hold` en la sentencia **open**.

### Conceptos relacionados

"resultSetID" en la página 743

"Soporte de SQL" en la página 229

### Tareas relacionadas

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

### Consulta relacionada

"delete" en la página 571

"execute" en la página 574

"get" en la página 585

"get absolute" en la página 591

"get current" en la página 593

"get first" en la página 594

"get next"

"get previous" en la página 602

"get relative" en la página 606

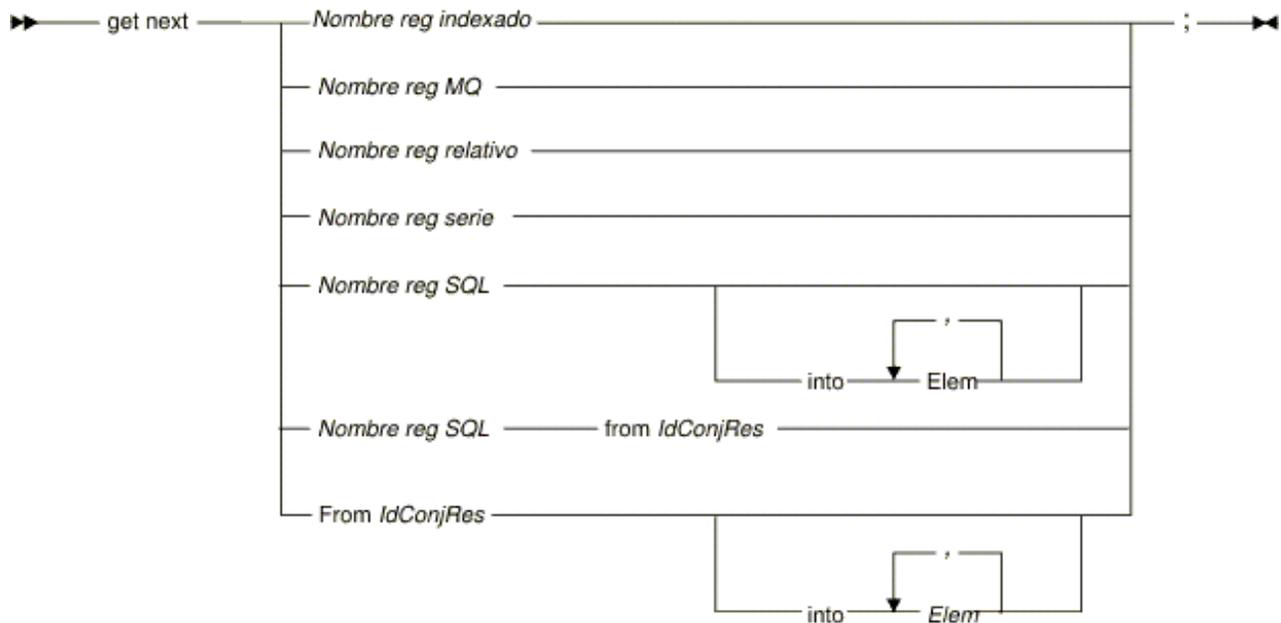
"Sentencias EGL" en la página 88

"open" en la página 616

"replace" en la página 632

## get next

La sentencia EGL **get next** lee el registro siguiente de un archivo o cola de mensajes, o la fila siguiente de una base de datos.



### nombre de registro

Nombre del objeto de E/S; un registro indexado, MQ, relativo, serie o SQL

### from IDconjuntoResultados

Sólo para procesos SQL, un ID que conecta la sentencia **get next** con una sentencia **open** ejecutada antes en el mismo programa. Para obtener detalles, consulte el apartado *resultSetID*.

### **into**

Inicia una cláusula EGL into, que lista los elementos que recibe valores de una tabla de base de datos relacional.

### *elemento*

Un elemento que recibe el valor de una columna determinada. *No* especifique un signo de dos puntos (:) ante el nombre del elemento.

A continuación se ofrece un ejemplo de acceso a archivo:

```
try
 open record1 forUpdate;
 onException
 myErrorHandler(8);
 return;
end
try
 get next record1;
 onException
 myErrorHandler(12);
 return;
end

while (record1 not endOfFile)
 makeChanges(record1); // procesar el registro

 try
 replace record1;
 onException
 myErrorHandler(16);
 return;
 end

 try
 get next record1;
 onException
 myErrorHandler(12);
 return;
 end
end // fin de while

sysLib.commit();
```

Los detalles de la sentencia **get next** dependen del tipo de registro. Para obtener detalles sobre el proceso SQL, consulte la sección “Proceso SQL” en la página 601.

## **Registro indexado**

Cuando una sentencia **get next** opera en un registro indexado, el resultado se basa en la posición actual del archivo, establecida mediante cualquiera de estas operaciones:

- Una operación de entrada o salida (E/S) satisfactoria, como por ejemplo una sentencia **get** u otra sentencia **get next**; o
- Una sentencia **set** en el formato *set record position*.

Las normas son las siguientes:

- Si el archivo no está abierto, la sentencia **get next** lee un registro con el valor de clave más bajo del archivo.
- Cada sentencia **get next** subsiguiente lee un registro cuyo valor de clave es el próximo más alto en relación a la posición actual del archivo. Más adelante se describe una excepción relativa a las claves duplicadas.

- Una vez que una sentencia **get next** ha leído el registro con el valor de clave más alto del archivo, la siguiente sentencia **get next** provocará el valor de error de E/S **endOfFile**.
- La posición actual del archivo resulta afectada por cualquiera de estas operaciones:
  - Una sentencia EGL **set** en el formato *set record position* establece una posición de archivo basada en el *valor de set*, que es el valor de clave del registro indexado al que la sentencia **set** hace referencia. La sentencia **get next** subsiguiente en el mismo registro indexado lee el registro de archivo cuyo valor de clave es igual o superior al valor de set. Si no existe tal registro, el resultado de la sentencia **get next** es **endOfFile**.
  - Una sentencia de E/S satisfactoria que no sea una sentencia **get next** establece una nueva posición de archivo y la sentencia **get next** subsiguiente emitida en el mismo registro EGL lee el *siguiente* registro del archivo. Una vez que una sentencia **get previous** ha leído un registro de archivo, por ejemplo, la sentencia **get next** lee el registro de archivo cuyo valor de clave es el siguiente más alto o devuelve **endOfFile**.
  - Si una sentencia **get previous** devuelve **endOfFile**, la sentencia **get next** subsiguiente recupera el primer registro del archivo.
  - Después de una sentencia **get**, **get next** o **get previous** no satisfactoria, la posición de archivo no está definida y debe volver a establecerse mediante una sentencia **set** en el formato *set record position* o mediante una operación de E/S que no sea una sentencia **get next** ni **get previous**.
- Si utiliza un índice alternativo y en el archivo hay claves duplicadas, se aplican las siguientes normas:
  - La recuperación de un registro con una clave de valor más alto sólo se produce después de que una sentencia **get next** haya leído todos los registros que tienen la misma clave como registro recuperado más recientemente. El orden en el que se recuperan los registros con claves duplicadas es el orden en el que VSAM devuelve los registros.
  - Si una sentencia **get next** sigue a una operación de E/S satisfactoria que no sea una sentencia **get next**, la sentencia **get next** pasa por alto los registros con claves duplicadas y recupera el registro con la siguiente clave más alta.
  - El valor de error EGL **duplicate** no se establece si el programa recupera el último registro de un grupo de registros que contienen la misma clave.

Considere un archivo en el que las claves son las siguientes:

1, 2, 2, 2, 3, 4

Cada una de las tablas siguientes ilustra el resultado de la ejecución de una secuencia de sentencias EGL en el mismo registro indexado.

Sentencia EGL (por orden)	Clave del registro indexado	Clave del registro de archivo recuperada por la sentencia	Valor de error EGL
<b>get</b>	2	2 (la primera de tres)	duplicate
<b>get next</b>	cualquiera	2 (la segunda)	duplicate
<b>get next</b>	cualquiera	2 (la tercera)	—
<b>get next</b>	cualquiera	3	—

Sentencia EGL (por orden)	Clave del registro indexado	Clave del registro de archivo recuperada por la sentencia	Valor de error EGL
<b>set</b> (en el formato <i>set record position</i> )	2	sin recuperación	duplicate
<b>get next</b>	cualquiera	2 (la primera de tres)	—
<b>get next</b>	cualquiera	2 (la segunda)	duplicate
<b>get next</b>	cualquiera	2 (la tercera)	—
<b>get next</b>	cualquiera	3	—

## Cola de mensajes

Cuando una sentencia **get next** opera en un registro MQ, el primer registro de la cola se lee en el registro MQ. Esta colocación se produce debido a que la sentencia **get next** invoca una o varias llamadas MQSeries:

- MQCONN conecta el código generado con el gestor de colas por omisión y se invoca cuando no hay ninguna conexión activa
- MQOPEN establece una conexión con la cola y se invoca cuando hay una conexión activa, pro la cola no está abierta
- MQGET elimina el registro de la cola y siempre se invoca, a menos que se haya producido un error en una llamada MQSeries anterior

## Registro relativo

Cuando una sentencia **get next** opera en un registro relativo, el resultado se basa en la posición actual del archivo, establecida mediante una operación de entrada o salida (E/S) satisfactoria, como por ejemplo una sentencia **get** u otra sentencia **get next**. Las normas son las siguientes:

- Si el archivo no está abierto, la sentencia **get next** lee el primer registro del archivo.
- Cada sentencia **get next** subsiguiente lee un registro cuyo valor de clave es el próximo más alto en relación a la posición actual del archivo.
- Una sentencia **get next** no devuelve **noRecordFound** si se suprime el registro siguiente. En lugar de ello, la sentencia **get next** pasa por alto los registros suprimidos y recupera el registro siguiente del archivo.
- Una vez que una sentencia **get next** ha leído el registro con el valor de clave más alto del archivo, la siguiente sentencia **get next** provocará el valor de error EGL **endOfFile**.
- La posición actual del archivo resulta afectada por cualquiera de estas operaciones:
  - Una sentencia de E/S satisfactoria que no sea una sentencia **get next** establece una nueva posición de archivo y la sentencia **get next** subsiguiente emitida en el mismo registro EGL lee el *siguiente* registro del archivo.
  - Después de una sentencia **get**, **get next** o **get previous** no satisfactoria, la posición de archivo no está definida y debe volver a establecerse mediante una sentencia **set** en el formato *set record position* o mediante una operación de E/S que no sea una sentencia **get next**.
- Una vez que una sentencia **get next** ha leído el último registro del archivo, la siguiente sentencia **get next** provocará los valores de error EGL **endOfFile** y **noRecordFound**.

## Registro serie

Cuando una sentencia **get next** opera en un registro serie, el resultado se basa en la posición actual del archivo, establecida mediante otra sentencia **get next**. Las normas son las siguientes:

- Si el archivo no está abierto, la sentencia **get next** lee el primer registro del archivo.
- Cada sentencia **get next** subsiguiente lee el registro siguiente.
- Una vez que una sentencia **get next** ha leído el último registro, la sentencia **get next** subsiguiente provocará el valor de error EGL **endOfFile**.
- Si el código generado añade un registro serie y luego emite el equivalente de una sentencia **get next** en el mismo archivo, EGL cierra y vuelve a abrir el archivo antes de ejecutar la sentencia **get next**. Por tanto, una sentencia **get next** que sigue a una sentencia **add** lee el primer registro del archivo. Este comportamiento también se produce cuando las sentencias **get next** y **add** se encuentran en programas diferentes, y un programa llama al otro.

Es aconsejable evitar que el mismo archivo esté abierto en más de un programa simultáneamente.

## Proceso SQL

Cuando una sentencia **get next** opera en un registro SQL, el código lee la fila siguiente de las seleccionadas por una sentencia **open**. Si emite una sentencia **get next** para recuperar una fila seleccionada mediante una sentencia **open** que tiene la opción **forUpdate**, puede realizar cualquiera de estas acciones:

- Cambiar la fila con una sentencia EGL **replace**
- Eliminar la fila con una sentencia EGL **delete**
- Cambiar o eliminar la fila con una sentencia EGL **execute**

Una sentencia SQL **FETCH** representa la sentencia EGL **get next** en el código generado. El formato de la sentencia SQL generada no puede cambiarse, excepto para establecer la cláusula **INTO**.

Si emite una sentencia **get next** que intenta acceder a una fila que esté más allá de la última fila seleccionada, se aplican las sentencias siguientes:

- No se copian datos del conjunto de resultados
- EGL establece el registro SQL (si lo hay) en **noRecordFound**
- Si la sentencia **open** relacionada incluía la opción **scroll**, el cursor permanece abierto con la posición del cursor inalterada.
- Si no ha establecido la opción, **scroll**, el cursor se cierra.

Finalmente, al especificar **SQL COMMIT** o **sysLib.commit**, el código conserva la posición en el cursor declarado en la sentencia **open**, pero sólo en caso de que utilice la opción **hold** en la sentencia **open**.

## Conceptos relacionados

“Tipos de registros y propiedades” en la página 135

“resultSetID” en la página 743

“Soporte de SQL” en la página 229

## Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

## Consulta relacionada

"add" en la página 561

"close" en la página 568

"delete" en la página 571

"Manejo de excepciones" en la página 94

"execute" en la página 574

"get" en la página 585

"get previous"

"Valores de error de E/S" en la página 536

"Sentencias EGL" en la página 88

"open" en la página 616

"prepare" en la página 630

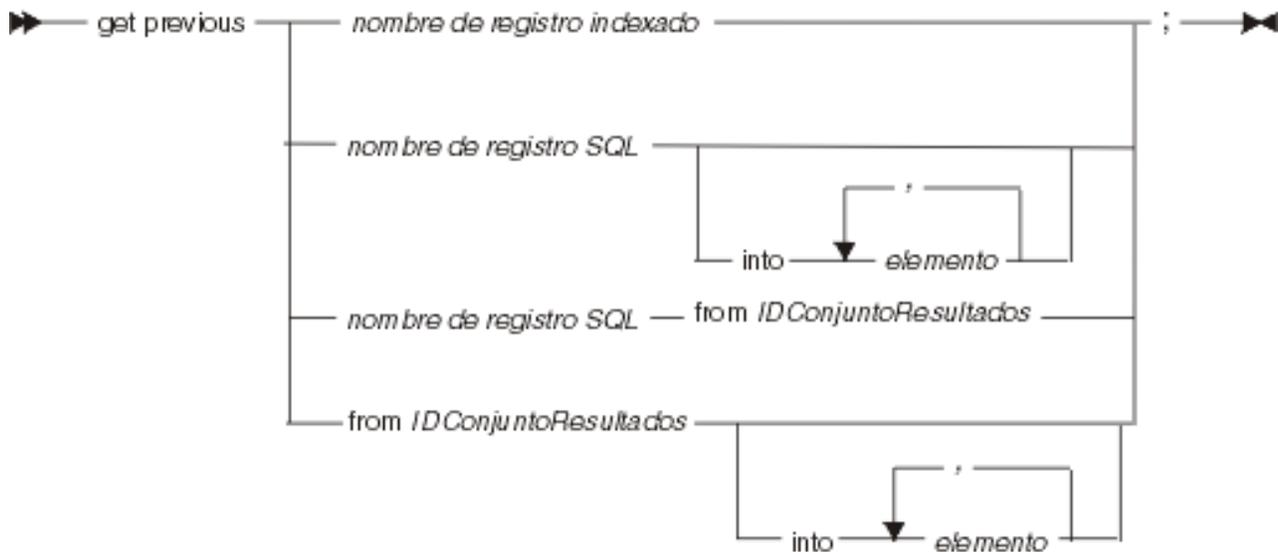
"replace" en la página 632

"set" en la página 636

## get previous

La sentencia EGL **get previous** o bien lee la fila anterior de un conjunto de resultados de base de datos relacional, o bien lee el registro anterior del archivo asociado con un registro indexado de EGL.

Solo puede utilizar esta sentencia para un conjunto de resultados de base de datos relacional si especificó la opción scroll en la sentencia **open** relacionada.



*nombre de registro*

Nombre del objeto de E/S; un registro indexado o SQL.

**from** *IDconjuntoResultados*

Sólo para procesos SQL, un ID que conecta la sentencia **get previous** con una sentencia **open** ejecutada antes en el mismo programa. Para obtener detalles, consulte el apartado *resultSetID*.

**into**

Inicia una cláusula EGL *into*, que lista los elementos que recibe valores de una tabla de base de datos relacional.

*elemento*

Un elemento que recibe el valor de una columna determinada. *No* especifique un signo de dos puntos (:) ante el nombre del elemento.

A continuación se ofrece un ejemplo de registro indexado:

```
record1.hexKey = "FF";
set record1 position;

try
 get previous record1;
onException
 myErrorHandler(8);
return;
end

while (record1 not endOfFile)
 processRecord(record1); // manejar los datos

 try
 get previous record1;
 onException
 myErrorHandler(8);
 return;
end
end
```

Los detalles de la sentencia **get previous** dependen de si está utilizando un registro indexado o de si está ocupando de "Proceso SQL" en la página 605.

### Registro indexado

Cuando una sentencia **get previous** opera en un registro indexado, el resultado se basa en la posición actual del archivo, establecida mediante cualquiera de estas operaciones:

- Una operación de entrada o salida (E/S) satisfactoria, como por ejemplo una sentencia **get** u otra sentencia **get previous**; o
- Una sentencia **set** en el formato *set record position*.

Las reglas para un registro indexado son las siguientes:

- Si el archivo no está abierto, la sentencia **get previous** lee un registro con el valor de clave más alto del archivo.
- Cada sentencia **get previous** subsiguiente lee un registro cuyo valor de clave es el próximo más bajo en relación a la posición actual del archivo. Más adelante se describe una excepción relativa a las claves duplicadas.
- Una vez que una sentencia **get previous** ha leído el registro con el valor de clave más bajo del archivo, la siguiente sentencia **get previous** provocará el valor de error EGL **endOfFile**.
- La posición actual del archivo resulta afectada por cualquiera de estas operaciones:
  - Una sentencia EGL **set** en el formato *set record position* establece una posición de archivo basada en el *valor de set*, que es el valor de clave del registro indexado al que la sentencia **set** hace referencia. La sentencia **get previous** subsiguiente en el mismo registro indexado lee el registro de archivo cuyo valor de clave es igual o inferior al valor de set. Si no existe tal registro, el resultado de la sentencia **get previous** es **endOfFile**.

Si el valor de set está relleno con caracteres FF hexadecimales, el resultado de una sentencia **set** en el formato *set record position* es el siguiente:

- La sentencia **set** establece una posición de archivo posterior al último registro del archivo
- Si la próxima operación de E/S es una sentencia **get previous**, el código generado recupera el último registro del archivo
- Una sentencia de E/S satisfactoria que no sea una sentencia **get previous** establece una nueva posición de archivo y la sentencia **get previous** subsiguiente emitida en el mismo registro EGL lee el registro de archivo *anterior*. Una vez que una sentencia **get next** ha leído un registro de archivo, por ejemplo, la sentencia **get previous** lee el registro de archivo cuyo valor de clave es el siguiente más bajo o devuelve **endOfFile**.
- Si una sentencia **get next** devuelve **endOfFile**, la sentencia **get previous** subsiguiente recupera el último registro del archivo.
- Después de una sentencia **get**, **get next** o **get previous** no satisfactoria, la posición de archivo no está definida y debe volver a establecerse mediante una sentencia **set** en el formato *set record position* o mediante una operación de E/S que no sea una sentencia **get next** ni **get previous**.
- Si utiliza un índice alternativo y en el archivo hay claves duplicadas, se aplican las siguientes normas:
  - La recuperación de un registro con una clave de valor más bajo sólo se produce después de que una sentencia **get previous** haya leído todos los registros que tienen la misma clave como registro recuperado más recientemente. El orden en el que se recuperan los registros con claves duplicadas es el orden en el que VSAM devuelve los registros.
  - Si una sentencia **get previous** sigue a una operación de E/S satisfactoria que no sea una sentencia **get previous**, la sentencia **get previous** pasa por alto los registros con claves duplicadas y recupera el registro con la siguiente clave más baja.
  - El valor de error EGL **duplicate** no se establece si el programa recupera el último registro de un grupo de registros que contienen la misma clave.

Considere un archivo en el que las claves de un índice alternativo son las siguientes:

1, 2, 2, 2, 3, 4

Cada una de las tablas siguientes ilustra el resultado de la ejecución de una secuencia de sentencias EGL en el mismo registro indexado.

Sentencia EGL (por orden)	Clave del registro indexado	Clave del registro de archivo recuperada por la sentencia	Valor de error EGL para COBOL
<b>set</b> (en el formato <i>set record position</i> )	1	--	--
<b>get previous</b>	cualquiera	1	--

Sentencia EGL (por orden)	Clave del registro indexado	Clave del registro de archivo recuperada por la sentencia	Valor de error EGL
<b>get</b>	3	3	—
<b>get previous</b>	cualquiera	2 (la primera de tres)	duplicate
<b>get previous</b>	cualquiera	2 (la segunda)	duplicate
<b>get previous</b>	cualquiera	2 (la tercera)	—
<b>get previous</b>	cualquiera	1	—

Sentencia EGL (por orden)	Clave del registro indexado	Clave del registro de archivo recuperada por la sentencia	Valor de error EGL
<b>set</b> (en el formato <i>set record position</i> )	2	—	duplicate
<b>get next</b>	cualquiera	2 (la primera)	—
<b>get next</b>	cualquiera	2 (la segunda)	duplicate
<b>get previous</b>	cualquiera	1	—
<b>get previous</b>	cualquiera	--	endOfFile

Sentencia EGL (por orden)	Clave del registro indexado	Clave del registro de archivo recuperada por la sentencia	Valor de error EGL
<b>set</b> (en el formato <i>set record position</i> )	1	--	--
<b>get previous</b>	cualquiera	1	--

## Proceso SQL

Cuando una sentencia **get previous** opera en un registro SQL, el código lee la fila anterior de las seleccionadas por una sentencia **open** pero solo si ha especificado la opción scroll. Si emite una sentencia **get previous** para recuperar una fila seleccionada mediante una sentencia **open** que también tiene la opción forUpdate, puede realizar cualquiera de estas acciones:

- Cambiar la fila con una sentencia EGL **replace**
- Eliminar la fila con una sentencia EGL **delete**
- Cambiar o eliminar la fila con una sentencia EGL **execute**

Una sentencia SQL FETCH representa la sentencia EGL **get previous** en el código generado. El formato de la sentencia SQL generada no puede cambiarse, excepto para establecer la cláusula INTO.

Si emite una sentencia **get previous** que intenta acceder a una fila anterior a la primera fila seleccionada, el tiempo de ejecución EGL actúa de la manera siguiente:

- No copia datos del conjunto de resultados
- Deja el cursor abierto, con la posición del cursor inalterada
- Establece el registro SQL (si lo hay) en **noRecordFound**

Por lo general, si se produce un error y el proceso continúa, el cursor permanece abierto, con la posición del cursor inalterada.

Finalmente, al especificar SQL COMMIT o sysLib.commit, el código conserva la posición en el cursor declarado en la sentencia **open**, pero sólo en caso de que utilice la opción hold en la sentencia **open**.

## Conceptos relacionados

“Tipos de registros y propiedades” en la página 135

## Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

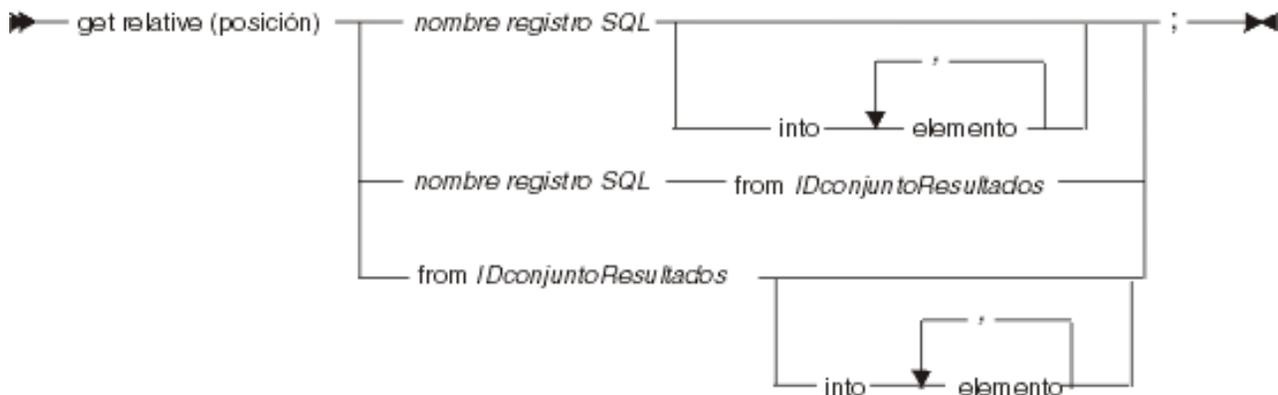
### Consulta relacionada

"add" en la página 561  
"close" en la página 568  
"delete" en la página 571  
"Manejo de excepciones" en la página 94  
"execute" en la página 574  
"get" en la página 585  
"get next" en la página 597  
"Valores de error de E/S" en la página 536  
"open" en la página 616  
"prepare" en la página 630  
"Sentencias EGL" en la página 88  
"replace" en la página 632  
"set" en la página 636

## get relative

La sentencia EGL **get relative** lee una fila especificada numéricamente en un conjunto de resultados de base de datos relacional. La fila se identifica en relación con la posición del cursor en el conjunto de resultados.

Solo puede utilizar esta sentencia si especificó la opción scroll en la sentencia **open** relacionada.



### *posición*

Literal o elemento entero.

Si el valor de *posición* es positivo, la posición es un incremento de la posición numérica actual en el conjunto de resultados. Si se especifica **get relative 2** cuando el cursor está en la primera fila, por ejemplo, se recupera la tercera fila y especificar **get relative 1** es equivalente a especificar **get next**.

Si el valor de *posición* es negativo, la posición es un decremento de la posición numérica actual en el conjunto de resultados. Si se especifica **get relative -2** cuando el cursor está en la tercera fila, por ejemplo, se recupera la primera fila y especificar **get relative -1** es equivalente a especificar **get previous**.

Un valor cero para *posición* recupera la fila en la posición del cursor que ya es efectiva y equivalente a especificar **get current**.

*nombre de registro*

Nombre de un registro SQL.

**from** *IDconjuntoResultados*

Un ID que conecta la sentencia **get relative** con una sentencia **open** ejecutada anteriormente en el mismo programa. Para obtener detalles, consulte el apartado *resultSetID*.

**into**

Inicia una cláusula EGL into, que lista los elementos que recibe valores de una tabla de base de datos relacional.

*elemento*

Un elemento que recibe el valor de una columna determinada. *No* especifique un signo de dos puntos (:) ante el nombre del elemento.

Si emite una sentencia **get relative** para recuperar una fila seleccionada mediante una sentencia **open** que tiene la opción forUpdate, puede realizar cualquiera de estas acciones:

- Cambiar la fila con una sentencia EGL **replace**
- Eliminar la fila con una sentencia EGL **delete**
- Cambiar o eliminar la fila con una sentencia EGL **execute**

Una sentencia SQL FETCH representa la sentencia EGL **get relative** en el código generado. El formato de la sentencia SQL generada no puede cambiarse, excepto para establecer la cláusula INTO.

Si emite una sentencia **get relative** que intente acceder a una fila que no esté en el conjunto de resultados, el tiempo de ejecución EGL actúa de la forma siguiente:

- No copia datos del conjunto de resultados
- Deja el cursor abierto con la posición del cursor inalterada
- Establece el registro SQL (si lo hay) en **noRecordFound**

Por lo general, si se produce un error y el proceso continúa, el cursor permanece abierto, con la posición del cursor inalterada.

Finalmente, al especificar SQL COMMIT o sysLib.commit, el código conserva la posición en el cursor declarado en la sentencia **open**, pero sólo en caso de que utilice la opción hold en la sentencia **open**.

#### **Conceptos relacionados**

"resultSetID" en la página 743

"Soporte de SQL" en la página 229

#### **Tareas relacionadas**

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

#### **Consulta relacionada**

"delete" en la página 571

"Manejo de excepciones" en la página 94

"execute" en la página 574

"get" en la página 585

"get absolute" en la página 591

"get current" en la página 593

"get first" en la página 594

"get last" en la página 595

“get next” en la página 597  
“get previous” en la página 602  
“Sentencias EGL” en la página 88  
“open” en la página 616  
“replace” en la página 632

## goTo

La sentencia EGL **goTo** hace que el proceso continúe en una etiqueta especificada, que debe estar en la misma función que la sentencia y fuera de un bloque.

► goto *etiq* :

---

### *etiqueta*

Una serie de caracteres que se visualiza en algún lugar de la función, fuera de los bloques, incluidos los siguientes:

- if
- else
- when (en una sentencia **case**)
- while
- try

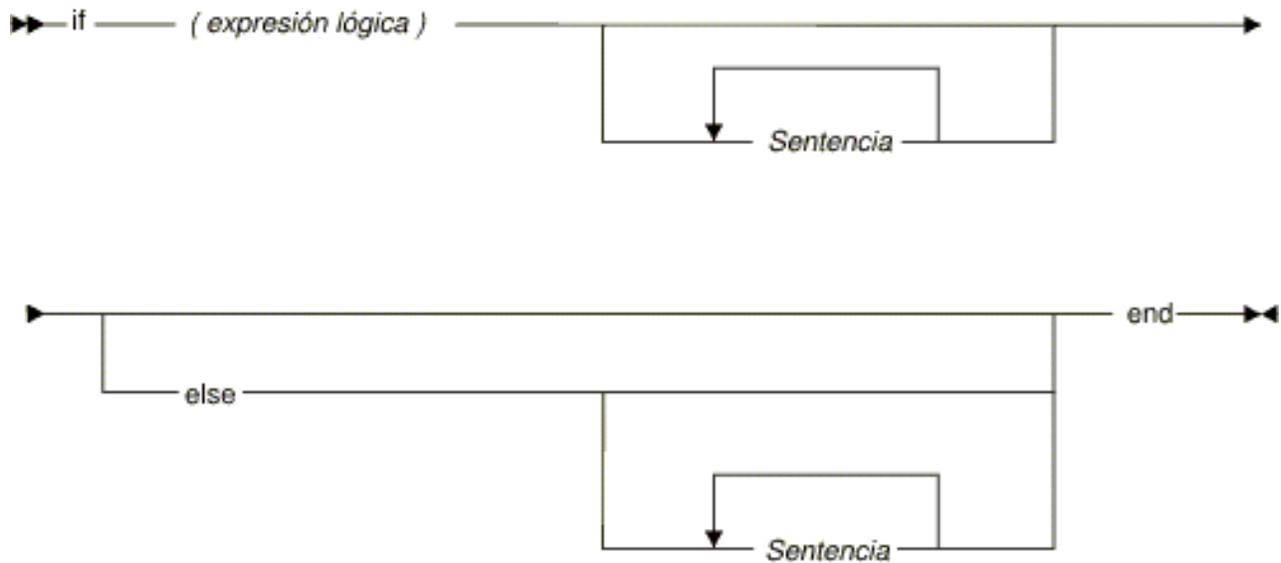
Cuando se visualiza en la ubicación en la que continúa el proceso, la etiqueta va seguida de un signo de dos puntos. Para obtener detalles acerca de los caracteres válidos para la etiqueta, consulte el apartado *Convenios de denominación*.

### **Consulta relacionada**

“Convenios de denominación” en la página 672

## if, else

La palabra clave EGL **if** marca el inicio de un conjunto de sentencias (si las hay) que sólo se ejecutan si una expresión lógica se resuelve en true. La palabra clave opcional **else** marca el inicio de un conjunto de sentencias alternativo (si las hay) que sólo se ejecutan si la expresión lógica se resuelve en false. La palabra clave *end* marca el cierre de la sentencia *if*.



*expresión lógica*

Una expresión (una serie de operandos y operadores) que evalúa en true o false

*sentencia*

Una o varias sentencias EGL

Puede anidar **if** y las demás sentencias terminadas en **end** a cualquier nivel. Cada palabra clave **end** hace referencia a la sentencia más reciente que no ha finalizado y que se inicia con una de estas palabras clave:

- **if**
- **case**
- **try**
- **while**

Ninguna de estas sentencias va seguida de un signo de punto y coma.

A continuación se ofrece un ejemplo:

```

if (userRequest == "U")
 try
 update myRecord;
 onException
 myErrorHandler(12); // finaliza el programa
 end
 try
 myRecord.myItem=25;
 replace record1;
 onException
 myErrorHandler(16);
 end
else
 try
 add record2;
 onException
 myErrorHandler(18); // finaliza el programa
 end
 if (sysVar.systemType is WIN)
 myFunction01();
 end
end

```

```

else
 myFunction02();
end
end

```

### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### Consulta relacionada

“Expresiones lógicas” en la página 497

“Sentencias EGL” en la página 88

## move

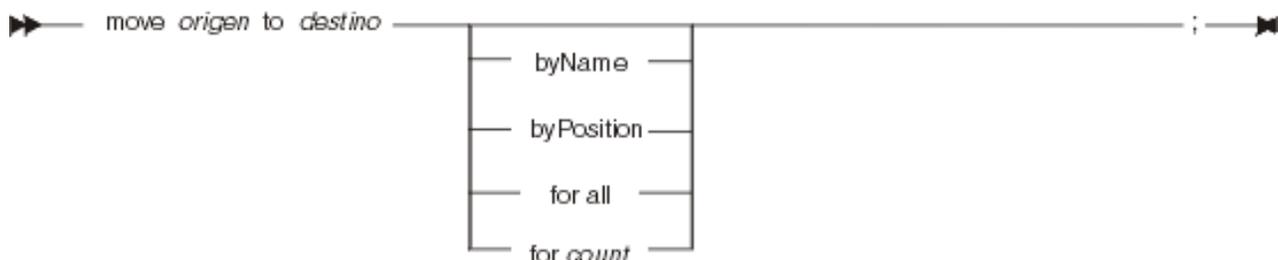
La sentencia EGL **move** copia datos de tres maneras distintas. La primera opción copia datos byte a byte, la segunda (llamada *por nombre*) copia datos de los campos con nombre en una estructura a los campos con el mismo nombre en otra y la tercera (llamada *por posición*) copia datos de cada campo de una estructura al campo de la posición equivalente en otra.

Se aplican las reglas generales siguientes:

- Si el valor origen es uno de estos, el valor predeterminado consiste en copiar datos byte a byte:
  - Una variable primitiva
  - Un campo que está en una estructura fija
  - Un literal
  - Una constante

De lo contrario, el valor predeterminado consiste en copiar datos por nombre.

- Se comprueba la compatibilidad campo a campo de cada movimiento. Las reglas de truncado, relleno y conversión de tipos son las mismas que las detalladas para la sentencia **assignment**, sin embargo, el comportamiento general de la sentencia **move** es diferente de la de la sentencia **assignment**.
- Si trabaja con matrices dinámicas, el último elemento queda determinado por el tamaño actual de la matriz. La sentencia **move** nunca añade un elemento de una matriz; para ampliar una matriz dinámica, utilice las funciones específicas de matriz `appendElement` o `appendAll`, tal como se describe en la sección *Matriz*.



La sentencia se entiende mejor en relación con las categorías siguientes:

### byName

Cuando especifica **byName**, se transcriben los datos de cada campo del origen a un campo con el mismo nombre del destino. La operación se produce en el orden de los campos en el origen.

A continuación se proporcionan ejemplos de origen y destino:

*origen*

Uno de los siguientes:

- Una matriz dinámica de registros fijos, pero la matriz solo es válida si el destino no es un registro
- Un registro
- Un registro fijo
- Un campo de estructura con una subestructura
- Una matriz de campo de estructura con una subestructura, pero esta matriz solo es válida si el destino no es un registro
- Una dataTable
- Un formulario

Un campo de estructura fija cuyo nombre sea un asterisco (\*) no está disponible como campo origen, pero sí están disponibles los campos con nombre de una subestructura de ese campo.

*destino*

Uno de los siguientes:

- Una matriz dinámica de registros fijos, pero esta matriz solo es válida si el origen no es un registro
- Un registro
- Un registro fijo
- Un campo de estructura con una subestructura
- Una matriz de campo de estructura con una subestructura, pero esta matriz solo es válida si el origen no es un registro
- Una dataTable
- Un formulario
- 

Una sentencia de ejemplo es la siguiente:

```
move myRecord01 to myRecord02 byName;
```

La operación no es válida en cualquiera de estos casos:

- Hay dos o más campos en el origen con el mismo nombre
- Hay dos o más campos en el destino con el mismo nombre
- El campo origen es una matriz de campo de estructura multidimensional o una matriz de campo de estructura unidimensional cuyo contenedor es una matriz
- El campo destino es una matriz de campo de estructura multidimensional o una matriz de campo de estructura unidimensional cuyo contenedor es una matriz

La operación funciona de la manera siguiente:

- En un caso simple, el origen es una estructura fija pero no es un elemento de matriz y lo mismo vale para el destino. Se aplican las reglas siguientes:
  - Si no hay matrices implicadas, el valor de cada campo subordinado en la estructura de origen se copia en el campo del mismo nombre de la estructura destino.
  - Si una matriz de campos de estructura se copia en una matriz de campos de estructura, la operación se trata como una operación *move for all*:

- Los elementos del campo origen se copian en elementos sucesivos del campo destino
- Si la matriz origen tiene menos elementos que la matriz destino, el proceso se detiene cuando se copia el último elemento de la matriz origen
- En otro caso, el origen o el destino es un registro. Los campos del origen se asigna a los campos del mismo nombre del destino.
- La mejor manera de presentar un caso menos simples es mediante un ejemplo. El origen es una matriz de 10 registros fijos, cada una de la cuál incluye estos campos de estructura:

```
10 empnum CHAR(3);
10 empname CHAR(20);
```

El destino es una estructura fija que incluye estos campos de estructura:

```
10 empnum CHAR(3)[10];
10 empname CHAR(20)[10];
```

La operación copia el valor del campo empnum del primer registro fijo en el primer elemento de la matriz de campo de estructura empnum, copia el valor del campo empname del primer registro fijo en el primer elemento de la matriz de campo de estructura empname y realiza una operación similar para cada registro fijo de la matriz origen.

La operación equivalente ocurre si el origen es un solo registro fijo que tiene una subestructura como esta:

```
10 mySubStructure[10]
15 empnum CHAR(3);
15 empname CHAR(20);
```

- Finalmente, observe el caso en el que el origen es un registro fijo que incluye estos campos de estructura:

```
10 empnum CHAR(3);
10 empname CHAR(20)[10];
```

El destino es un formulario, un registro fijo o un campo de estructura que tiene la subestructura siguiente:

```
10 empnum char(3)[10];
10 empname char(20);
```

El valor del campo empnum se copia del origen al primer elemento de empnum en el destino y el valor del primer elemento de empname se copia del origen al campo empname en el destino.

### **byPosition**

El propósito de **byPosition** es copiar datos de cada campo en una estructura en el campo de la posición equivalente en otra.

A continuación se proporcionan ejemplos de origen y destino:

*origen*

Uno de los siguientes:

- Una matriz dinámica de registros fijos, pero la matriz solo es válida si el destino no es un registro
- Un registro
- Un registro fijo
- Un campo de estructura con una subestructura
- Una matriz de campo de estructura con una subestructura, pero esta matriz solo es válida si el destino no es un registro
- Una dataTable
- Un formulario

*destino*

Uno de los siguientes:

- Una matriz dinámica de registros fijos, pero esta matriz solo es válida si el origen no es un registro
- Un registro
- Un registro fijo
- Un campo de estructura con una subestructura
- Una matriz de campo de estructura con una subestructura, pero esta matriz solo es válida si el origen no es un registro
- Una dataTable
- Un formulario

Cuando mueve datos entre un registro y una estructura fija, solo se tienen en cuenta los campos de nivel superior de la estructura fija. Cuando mueve los datos entre dos estructuras fijas, solo se tienen en cuenta los campos de nivel inferior (hoja) de cada estructura.

La operación no es válida si el campo origen o destino es una matriz de campo de estructura multidimensional o una matriz de campo de estructura unidimensional cuyo contenedor sea una matriz.

La operación funciona de la manera siguiente:

- En un caso simple, el origen es una estructura fija pero no es un elemento de matriz y lo mismo vale para el destino. Se aplican las reglas siguientes:
  - Si no hay matrices implicadas, el valor de cada campo de hoja de la estructura origen se copia en el campo de hoja de la estructura destino en la posición correspondiente.
  - Si una matriz de campos de estructura se copia en una matriz de campos de estructura, la operación se trata como una operación *move for all*:
    - Los elementos del campo origen se copian en elementos sucesivos del campo destino
    - Si la matriz origen tiene menos elementos que la matriz destino, el proceso se detiene cuando se copia el último elemento de la matriz origen
- En otro caso, el origen o el destino es un registro. Los campos de nivel superior o campos de hoja del origen (dependiendo del tipo de origen) se asignan a los campos de nivel superior o campos de hoja del destino (dependiendo del tipo de destino).
- La mejor manera de presentar un caso menos simple es mediante un ejemplo. El origen es una matriz de 10 registros fijos, cada uno de los cuales incluye estos campos de estructura:

```
10 empnum CHAR(3);
10 empname CHAR(20);
```

El destino es una estructura fija que incluye estos campos de estructura:

```
10 empnum CHAR(3)[10];
10 empname CHAR(20)[10];
```

La operación copia el valor del campo empnum del primer registro fijo en el primer elemento de la matriz de campo de estructura empnum, copia el valor del campo empname del primer registro fijo en el primer elemento de la matriz de campo de estructura empname y realiza una operación similar para cada registro fijo de la matriz origen.

La operación equivalente ocurre si el origen es un solo registro fijo que tiene una subestructura como esta:

```
10 mySubStructure[10]
 15 empnum CHAR(3);
 15 empname CHAR(20);
```

- Finalmente, observe el caso en el que el origen es un registro fijo que incluye estos campos de estructura:

```
10 empnum CHAR(3);
10 empname CHAR(20)[10];
```

El destino es un formulario, un registro fijo o un campo de estructura que tiene la subestructura siguiente:

```
10 empnum char(3)[10];
10 empname char(20);
```

El valor del campo empnum se copia del origen al primer elemento de empnum en el destino y el valor del primer elemento de empname se copia del origen al campo empname en el destino.

### **for all**

El propósito de **for all** es asignar valores a todos los elementos de una matriz destino.

A continuación se proporcionan ejemplos de origen y destino:

#### *origen*

Uno de los siguientes:

- Una matriz dinámica de registros, registros fijos o variables primitivas
- Un registro
- Un registro fijo
- Un campo de estructura con o sin subestructura
- Una matriz de campo de estructura con o sin subestructura
- Una variable primitiva
- Un literal o constante

#### *destino*

Uno de los siguientes:

- Una matriz dinámica de registros, registros fijos o variables primitivas
- Una matriz de campo de estructura con o sin subestructura
- Un elemento de una matriz dinámica o de campo de estructura

La sentencia **move** en este caso es equivalente a varias sentencias **assignment**, una por elemento de matriz destino y se produce un error si una asignación intentada no es válida. Para obtener detalles sobre la validez, consulte la sección *Asignaciones*.

Si un elemento origen o destino tiene una estructura fija, la sentencia **move** trata esa estructura como un campo de tipo CHAR a menos que el nivel superior de la estructura especifique un tipo primitivo distinto. Cuando se esté utilizando **for all**, la sentencia **move** no tiene en cuenta la subestructura.

Si el origen es un elemento de una matriz, el origen se trata como una matriz en la que el elemento especificado es el primer elemento y se ignoran los elementos anteriores.

Si el origen es una matriz o un elemento de una matriz, cada elemento sucesivo de la matriz origen se copia en el elemento siguiente según la

secuencia de la matriz destino. La matriz destino o la matriz origen pueden ser una mayor que la otra y la operación finaliza cuando se copian datos del último elemento con un elemento coincidente en la otra matriz.

Si el origen no es una matriz ni un elemento de una matriz, la operación utiliza el valor origen para inicializar cada elemento de la matriz destino.

### **for cuenta**

El propósito de **for cuenta** es asignar valores a un subconjunto secuencial de elementos de una matriz destino. Ejemplos:

- La sentencia siguiente mueve "abc" a los elementos 7, 8 y 9 en el destino:  
`move "abc" to target[7] for 3`
- La sentencia siguiente mueve los elementos 2, 3 y 4 del origen a los elementos 7, 8 y 9 del destino:  
`move source[2] to target[7] for 3`

La operación funciona de la manera siguiente:

- Si el origen no es una matriz ni un elemento de una matriz, la operación utiliza el valor origen para inicializar elementos de la matriz destino.
- Si el origen es una matriz, el primer elemento de esa matriz es el primero de un conjunto de elementos a copiar. Si el origen es un elemento de una matriz, ese elemento es el primero de un conjunto de elementos a copiar.
- Si el destino es una matriz, el primer elemento de esa matriz es el primero de un conjunto de elementos en recibir datos. Si el destino es un elemento de una matriz, ese elemento es el primero de un conjunto de elementos que recibe datos.

El valor *cuenta* indica cuántos elementos destino deben recibir datos. El valor puede ser cualquiera de estos:

- Un literal entero
- Una variable que se resuelve en un entero
- Un expresión numérica, pero no una invocación de función

La sentencia **move** es equivalente a varias sentencias **assignment**, una por elemento de matriz destino y se produce un error si una asignación intentada no es válida. Para obtener detalles sobre la validez, consulte la sección *Asignaciones*.

Si un elemento origen o destino tiene una estructura interna, la sentencia **move** trata esa estructura como un campo de tipo CHAR a menos que el nivel superior de esa estructura especifique un tipo primitivo distinto. Cuando se está utilizando **for cuenta**, la sentencia **move** no tiene en cuenta la subestructura.

Cuando el origen y el destino son matrices, la matriz destino o la matriz origen pueden ser una mayor que la otra y la operación finaliza cuando tiene lugar el primero de dos sucesos:

- Se copian datos entre los últimos elementos para los que se ha solicitado la operación
- Se copian datos desde el último elemento con un elemento coincidente en la otra matriz.

Cuando el origen no es una matriz, la operación finaliza cuando tienen lugar el primero de dos sucesos:

- Se copian datos en el último elemento para el que se ha solicitado la operación
- Se copian datos en el último elemento de la matriz.

Si un registro es una matriz de registros (o un elemento de uno), el destino debe ser una matriz de registros. Si el origen es una matriz de variable primitiva (o un elemento de una), el destino debe ser una matriz de variable primitiva o una matriz de campo de estructura. Si el origen es una matriz de campo de estructura (o un elemento de una), el destino debe ser una matriz de variable primitiva o una matriz de campo de estructura.

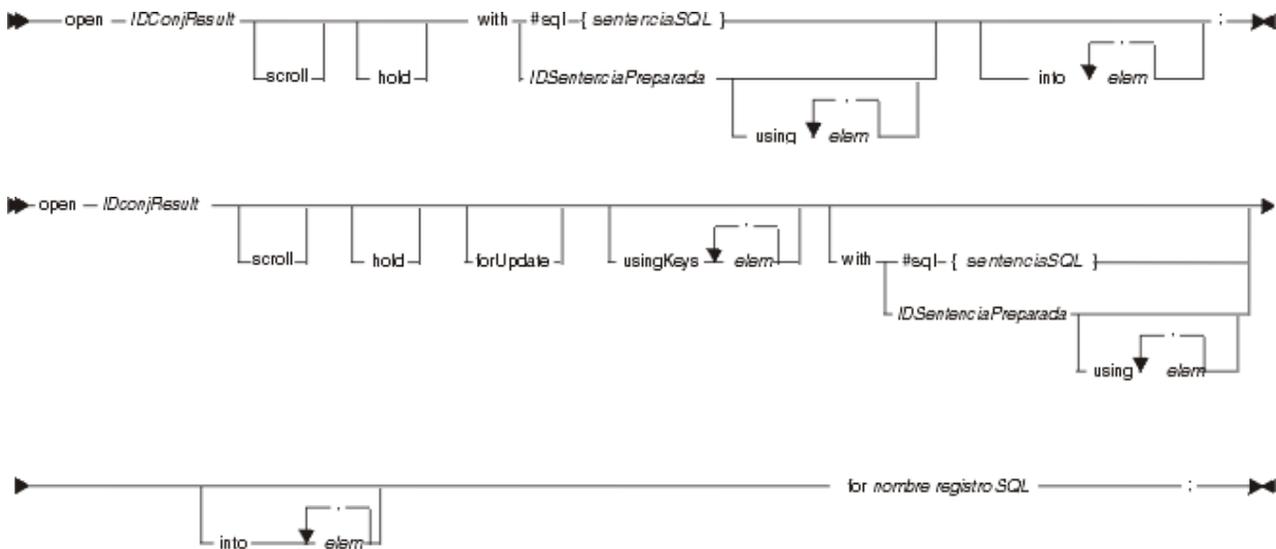
### Consulta relacionada

“Matrices” en la página 74

“Asignaciones” en la página 374

## open

La sentencia EGL **open** selecciona un conjunto de filas de una base de datos relacional para recuperarlo posteriormente mediante sentencias **get next**. La sentencia **open** puede operar sobre un cursor o sobre un procedimiento al que se llama.



### IDconjuntoResultados

ID que conecta la sentencia **open** con las sentencias **get next**, **replace**, **delete** y **close** posteriores. Para obtener detalles, consulte el apartado *resultSetID*.

### scroll

Opción que permite moverse a través de un conjunto de resultados de varias maneras. La sentencia **get next** siempre está disponible, pero la utilización de **scroll** permite utilizar también las sentencias siguientes:

- **get absolute**
- **get current**
- **get first**
- **get last**
- **get previous**
- **get relative**

## **hold**

Mantiene la posición en un conjunto de resultados cuando se produce una operación de compromiso.

**Nota:** La opción **hold** solo está disponible si el controlador JDBC soporta JDBC 3.0 o una versión superior.

La opción **hold** es adecuada en el siguiente caso:

- Está utilizando la sentencia EGL **open** para abrir un cursor en lugar de un procedimiento almacenado; y
- Desea comprometer los cambios periódicamente sin perder la posición en el conjunto de resultados; y
- El sistema de gestión de bases de datos da soporte a la utilización de la opción WITH HOLD en la declaración de cursores SQL.

El código podría hacer lo siguiente, por ejemplo:

1. Declarar una apertura de cursor ejecutando una sentencia EGL **open**
2. Extraer una fila ejecutando una sentencia EGL **get next**
3. Hacer lo siguiente en un bucle:
  - a. Procesar los datos de alguna forma
  - b. Actualizar la fila ejecutando una sentencia EGL **replace**
  - c. Comprometer los cambios ejecutando la función de sistema `sysLib.commit`
  - d. Extraer otra fila ejecutando una sentencia EGL **get next**

Si no especifica **hold**, la primera ejecución de paso 3d fallará debido a que el cursor ya no está abierto.

Los cursores para los que especifique **hold** no se cierran en una operación de compromiso, pero una operación de retrotracción o conexión a base de datos cierra todos los cursores.

Si no necesita conservar la posición del cursor durante una operación de compromiso, no especifique **hold**.

## **forUpdate**

Opción que permite utilizar una sentencia EGL posterior para sustituir o suprimir los datos recuperados de la base de datos.

No puede especificar **forUpdate** si está llamando a un procedimiento almacenado para recuperar un conjunto de resultados.

## **usingKeys ... elemento**

Identifica una lista de elementos de clave utilizados para construir el componente de clave-valor de la cláusula WHERE en la sentencia SQL implícita. La sentencia SQL implícita se utiliza durante la ejecución si no especifica una sentencia SQL explícita.

Si no especifica una cláusula **usingKeys**, el componente de clave-valor de la sentencia implícita se basa en el componente de registro SQL al que se hace referencia en la sentencia **open**.

La información de **usingKeys** se pasa por alto si especifica una sentencia SQL explícita.

**with #sql{ *sentencia SQL* }**

Una sentencia SQL SELECT explícita, que es opcional si también especifica un registro SQL. No deje espacios entre #sql y el corchete de apertura.

**into ... *elemento***

Una cláusula INTO que identifica las variables de lenguaje principal EGL que reciben valores desde el cursor o procedimiento almacenado. En una cláusula como esta (que está fuera de un bloque #sql{ }), no incluya un punto y coma antes del nombre de una variable de lenguaje principal.

**with ID*sentenciaPreparada***

El identificador de una sentencia EGL **prepare** que prepara una sentencia SQL SELECT o CALL durante la ejecución. La sentencia **open** ejecuta la sentencia SQL SELECT o CALL dinámicamente. Para obtener detalles, consulte el apartado *prepare*.

**using ... *elemento***

Una cláusula USING que identifica las variables de lenguaje principal EGL que quedan a disposición de la sentencia SQL SELECT o CALL durante la ejecución. En una cláusula como esta (que está fuera de un bloque #sql{ }), no incluya un punto y coma antes del nombre de una variable de lenguaje principal.

*nombre de registro SQL*

Nombre de un registro de tipo SQLRecord. Es obligatorio el nombre de registro o un valor para *sentenciaSQL*; si se omite *sentenciaSQL*, la sentencia SQL SELECT se deriva del registro SQL.

A continuación se ofrecen ejemplos (suponiendo la existencia de un registro SQL denominado *emp*):

```
open empSetId forUpdate for emp;

open x1 with
#sql{
 select empnum, empname, empphone
 from employee
 where empnum >= :empnum
 for update of empname, empphone
}

open x2 with
#sql{
 select empname, empphone
 from employee
 where empnum = :empnum
}
for emp;

open x3 with
#sql{
 call aResultSetStoredProc(:argumentItem)
}
```

## Proceso por omisión

Por omisión, el resultado de una sentencia open es el siguiente, cuando se especifica un registro SQL:

- La sentencia open hace disponible un conjunto de filas. Cada columna de las filas seleccionadas está asociada con un elemento de estructura y, excepto las columnas que están asociadas con un elemento de estructura de sólo lectura, todas las columnas están a disposición de la actualización subsiguiente mediante una sentencia EGL replace.

- Si sólo declara un elemento de clave para el registro SQL, la sentencia **open** selecciona todas las filas que cumplen la **condición de selección por omisión** específica del registro, siempre y cuando el valor de la columna de clave de tabla SQL sea igual o superior al valor del elemento de clave del registro SQL.
- Si se declaran varias claves para el registro SQL, la **condición de selección por omisión** específica del registro es el único criterio de búsqueda y la sentencia **open** recupera todas las filas que cumplen ese criterio.
- Si no especifica una clave de registro ni una condición de selección por omisión, la sentencia **open** selecciona todas las filas de la tabla.
- Las filas seleccionadas no se ordenan.

La sentencia EGL **open** está representada en el código generado por una declaración de cursor que incluye una sentencia SQL SELECT o SELECT FOR UPDATE. Por omisión, se cumple lo siguiente:

- La cláusula FOR UPDATE (si existe) no incluye elementos de estructura que sean de sólo lectura
- La sentencia SQL SELECT de un registro determinado es similar a la sentencia siguiente:

```
SELECT columna01,
 columna02, ...
 columnaNN
INTO :elementoRegistro01,
 :elementoRegistro02, ...
 :elementoRegistroNN
FROM nombreTabla
WHERE columnaClave01 = :elementoClave01
FOR UPDATE OF
 columna01,
 columna02, ...
 columnaNN
```

Puede alterar temporalmente el valor por omisión especificando una sentencia SQL en la sentencia EGL **open**.

## Condiciones de error

Existen diversas condiciones que no son válidas, incluidas las siguientes:

- Incluye una sentencia SQL a la que le falta una cláusula obligatoria para SELECT; las cláusulas obligatorias son SELECT, FROM y (si especifica **forUpdate**) FOR UPDATE OF
- El registro SQL está asociado con una columna que no existe durante la ejecución o que es incompatible con el elemento de estructura relacionado
- Especifica la opción **forUpdate** y el código intenta ejecutar una sentencia **open** en uno de los siguientes registros SQL:
  - Un registro SQL cuyos únicos elementos de estructura son de sólo lectura; o
  - Un registro SQL que está relacionado con más de una tabla SQL.

También surge un problema en el siguiente caso:

1. Personaliza una sentencia EGL **open** para actualización, pero no puede indicar que una columna de tabla SQL determinada está disponible para actualización; y
2. La sentencia **replace** que está relacionada con la sentencia **open** intenta revisar la columna.

Puede resolver este problema de cualquiera de estas formas:

- Al personalizar la sentencia EGL **open**, incluya el nombre de columna en la sentencia SQL SELECT, cláusula FOR UPDATE OF; o
- Al personalizar la sentencia EGL **replace**, elimine la referencia a la columna en la sentencia SQL UPDATE, cláusula SET; o
- Acepte los valores por omisión para las sentencias **open** y **replace**.

### Conceptos relacionados

“Tipos de registros y propiedades” en la página 135  
 “Soporte de SQL” en la página 229  
 “resultSetID” en la página 743  
 “Referencias a componentes” en la página 21

### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

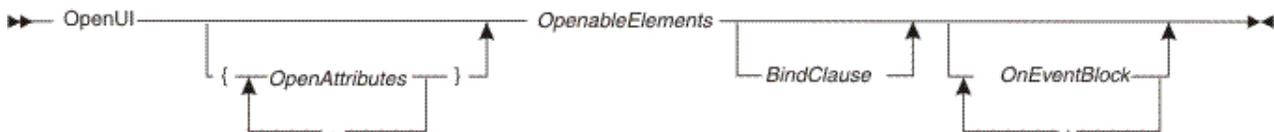
### Consulta relacionada

“add” en la página 561  
 “close” en la página 568  
 “delete” en la página 571  
 “Sentencias EGL” en la página 88  
 “Manejo de excepciones” en la página 94  
 “execute” en la página 574  
 “get” en la página 585  
 “get next” en la página 597  
 “get previous” en la página 602  
 “Valores de error de E/S” en la página 536  
 “prepare” en la página 630  
 “replace” en la página 632  
 “Propiedades de elementos SQL” en la página 67  
 “terminalID” en la página 938

## openUI

La sentencia **OpenUI** permite al usuario interactuar con un programa cuya interfaz se basa en consoleUI. La sentencia define eventos de usuario y programa y especifica cómo responder a cada uno de ellos.

La sintaxis de la sentencia OpenUI es la siguiente:



### OpenAttributes

*OpenAttributes* define un conjunto de pares de propiedad y valor, cada uno separado del siguiente por una coma, como en este ejemplo:

```
allowAppend = yes, allowDelete = no
```

Cada una de las propiedades afecta a la interacción de usuario y en algunos casos sobrescribe una propiedad de la variable consoleUI a la que se hace referencia en *OpenableElements*. Las propiedades son las siguientes:

## allowAppend

Especifica si el usuario puede insertar datos al final de un arrayDictionary en pantalla; si el valor es *yes*, las implicaciones son las siguientes:

- El usuario inserta una fila de datos moviendo el cursor a la línea arrayDictionary que sigue a la última línea que incluye datos
- La acción del usuario añade un elemento al final de la matriz dinámica que está enlazada a dicho arrayDictionary

**Para el tipo de variable:** *ArrayDictionary*

**Tipo de propiedad:** *Boolean*

**Ejemplo:** *allowAppend = no*

**Valor por omisión:** *yes; pero el valor por omisión es no si la propiedad de openUI displayOnly está establecida en yes*

## allowDelete

Especifica si el usuario puede suprimir una fila de un arrayDictionary en pantalla; si el valor es *yes*, las implicaciones son las siguientes:

- El usuario suprime una fila moviendo el cursor a dicha fila y pulsando la tecla a la que se hace referencia en **ConsoleLib.key\_delete**.
- La acción del usuario suprime el elemento relacionado de la matriz dinámica que está enlazada a dicho arrayDictionary.

**Para el tipo de variable:** *ArrayDictionary*

**Tipo de propiedad:** *Boolean*

**Ejemplo:** *allowDelete = no*

**Valor por omisión:** *yes; pero el valor por omisión es no si la propiedad de openUI displayOnly está establecida en yes*

## allowInsert

Especifica si el usuario puede insertar una fila en un arrayDictionary en pantalla; si el valor es *yes*, las implicaciones son las siguientes:

- El usuario inserta la fila moviendo el cursor a una ficha existente y pulsando la tecla a la que se hace referencia en **ConsoleLib.key\_insert**.
- La fila nueva precede a la fila que muestra el cursor
- La acción del usuario inserta un elemento en la matriz dinámica que está enlazada a dicho arrayDictionary.

**Para el tipo de variable:** *ArrayDictionary*

**Tipo de propiedad:** *Boolean*

**Ejemplo:** *allowInsert = no*

**Valor por omisión:** *yes; pero el valor por omisión es no si la propiedad de openUI displayOnly está establecida en yes*

## bindingByName

Indica cómo enlazar una serie de variables con una serie de ConsoleFields; concretamente, si se debe emparejar cada nombre de variable con un nombre de ConsoleField. El nombre de variable se lista en *BindClause* y el nombre de ConsoleField es el valor del campo Nombre de ConsoleField.

**Para el tipo de variable:** *ConsoleForm, ConsoleField o Dictionary; pero no arrayDictionary*

**Tipo de propiedad:** *Boolean*

**Ejemplo:** *bindByName = yes*

**Valor predeterminado:** *no*

Los valores son los siguientes:

**no (valor predeterminado)**

Emparejar variables y ConsoleFields por posición:

- La posición de cada variable en la lista; y
- La posición de cada ConsoleField en el consoleForm.

Independientemente de que los consoleFields se listen explícitamente en la sentencia openUI o se listen en una declaración dictionary, su orden define el orden de los consoleFields con la finalidad de enlazar por posición. (Su orden también define el orden de tabulación para la entrada del usuario, como se ha indicado en el apartado *Componentes ConsoleUI y variables relacionadas.*)

**yes**

Emparejar variables y ConsoleFields por nombre.

Si un consoleField se lista o se encuentra en una declaración dictionary cuando en la lista de enlaces no hay ninguna variable coincidente, la entrada del usuario en el consoleField se pasa por alto. De forma parecida, se pasa por alto una variable de enlace que no coincida con ningún campo.

Como mínimo un consoleField y una variable deben estar enlazados durante la ejecución; de lo contrario, se produce un error.

**color**

Especifica el color del texto en los ConsoleFields. El valor altera temporalmente el color especificado en la declaración de ConsoleField.

**Para el tipo de variable:** *ConsoleForm, ConsoleField, ArrayDictionary o Dictionary*

**Tipo de propiedad:** *ColorKind*

**Ejemplo:** *color = red*

**Valor predeterminado:** *white*

Los valores son los siguientes:

**defaultColor o white (valor predeterminado)**

Blanco

**black**

Negro

**blue**

Azul

**cyan**

Cian

**green**

Verde

**magenta**

Magenta

**red**

Rojo

**yellow**

Amarillo

### **currentArrayCount**

Especifica el número de elementos que están disponibles en la matriz dinámica a la que está enlazado el `arrayDictionary` en línea. Si no se especifica este valor, todos los elementos pueden utilizarse en el `arrayDictionary`.

**Para el tipo de variable:** *ArrayDictionary*

**Tipo de propiedad:** *INT*

**Ejemplo:** *currentArrayCount = 4*

**Valor predeterminado:** *none*

### **displayOnly**

Especifica si los `consoleFields` se visualizan sólo para verlos. Si el valor es *yes*, el usuario no puede modificar los datos, que están protegidos contra actualizaciones.

**Para el tipo de variable:** *ArrayDictionary, Dictionary, ConsoleField, ConsoleForm*

**Tipo de propiedad:** *Boolean*

**Ejemplo:** *displayOnly = yes*

**Valor predeterminado:** *no*

### **help**

Especifica el texto que debe visualizarse cuando el usuario pulsa la tecla identificada en `ConsoleLib.key_help`.

Este texto de ayuda está destinado al mandato **openUI**. En algunos casos, el texto asociado con la tecla es más específico del contexto. Por ejemplo, cada opción de un menú puede tener su propio mensaje de ayuda.

**Para el tipo de variable:** *ConsoleForm, ConsoleField, ArrayDictionary o Dictionary*

**Tipo primitivo:** *String*

**Ejemplo:** *help = "Actualizar el valor"*

**Valor predeterminado:** *Serie vacía*

### **helpKey**

Especifica una tecla de acceso para buscar el paquete de recursos que contiene el texto que debe visualizarse cuando se produce la siguiente situación:

- El cursor está en una variable de `ConsoleUI` (como por ejemplo, `ConsoleForm`) que se identifica en *OpenableElements*; y
- El usuario pulsa la tecla identificada en `ConsoleLib.key_help`.

Si se especifica a la vez **help** y **helpKey**, se utiliza **help**.

**Para el tipo de variable:** *ConsoleForm, ConsoleField, ArrayDictionary o Dictionary*

**Tipo de propiedad:** *String*

**Ejemplo:** *helpKey = "myKey"*

**Valor predeterminado:** *Serie vacía*

El paquete de recursos se identifica mediante la variable de sistema `ConsoleLib.messageResource`, tal como se describe en *messageResource*.

### **highlight**

Especifica los efectos especiales (si los hay) que se utilizan al visualizar el `ConsoleField`. El valor altera temporalmente el valor equivalente especificado en la declaración de `ConsoleField`.

**Para el tipo de variable:** *ConsoleForm, ConsoleField, ArrayDictionary o Dictionary*

**Tipo de propiedad:** *HighlightKind[]*

**Ejemplo:** *highlight = [reverse, underline]*

**Valor predeterminado:** *[noHighLight]*

Los valores son los siguientes:

**noHighlight (valor predeterminado)**

No produce ningún efecto especial. La utilización de este valor altera temporalmente cualquier otro.

**blink**

No tiene ningún efecto

**reverse**

Invierte los colores del texto y del fondo, de forma que (por ejemplo), si la pantalla tiene un fondo negro con letras blancas, el fondo pasa a ser negro y el texto pasa a ser blanco.

**underline**

Coloca un subrayado debajo de las áreas afectadas. El color del subrayado es el color del texto, aunque también se haya especificado el valor **reverse**.

**intensity**

Especifica la fuerza del font visualizado.

**Para el tipo de variable:** *ConsoleField, ConsoleForm, ArrayDictionary o Dictionary*

**Tipo de propiedad:** *IntensityKind[]*

**Ejemplo:** *intensity = [bold]*

**Valor predeterminado:** *[normalIntensity]*

Los valores son los siguientes:

**normalIntensity (valor predeterminado)**

No produce ningún efecto especial. La utilización de este valor altera temporalmente cualquier otro.

**bold**

Hace que el texto aparezca en negrita.

**dim**

Hace que el texto aparezca con menor intensidad, según sea apropiado cuando un campo de entrada está inhabilitado.

**invisible**

Elimina cualquier indicación de que el ConsoleField se encuentra en el formulario.

**isConstruct**

Especifica si la finalidad de la sentencia **openUI** es crear criterios de selección que se utilizarán en una sentencia SQL, como por ejemplo SELECT.

**Para el tipo de variable:** *ConsoleField, ConsoleForm, Dictionary*

**Tipo de propiedad:** *Boolean*

**Ejemplo:** *isConstruct = no*

**Valor predeterminado:** *yes*

Los valores son los siguientes:

**no (valor predeterminado)**

Cada ConsoleField está enlazado a una variable, como de costumbre.

**yes**

La sentencia **openUI** debe enlazarse a una única variable de un tipo de carácter. Dicha variable no proporciona valores iniciales para los ConsoleFields, pero sí recibe la entrada del usuario, que está formateada para utilizarla en una cláusula SQL WHERE.

**maxArrayCount**

Especifica el número máximo de filas que puede haber en la matriz dinámica enlazada al arrayDictionary en pantalla. Una vez que se ha alcanzado el máximo, el usuario no puede insertar más filas.

**Para el tipo de variable:** *ArrayDictionary*

**Tipo de propiedad:** *INT*

**Ejemplo:** *maxArrayCount = 20*

**Valor predeterminado:** *none*

**setInitial**

Especifica si se visualiza el valor inicial de un ConsoleField (según lo definido en la declaración de consoleForm) hasta que el usuario modifica dicho valor. (Especifique el valor inicial estableciendo el campo **initialValue** de ConsoleField.)

**Para el tipo de variable:** *ConsoleField, ConsoleForm, Dictionary, ArrayDictionary*

**Tipo de propiedad:** *Boolean*

**Ejemplo:** *setInitial = yes*

**Valor predeterminado:** *no*

Si el valor de **setInitial** es **no**, los valores de las variables enlazadas se extraen y visualizan inicialmente.

*OpenableElements*

Las variables de ConsoleUI sobre las que la sentencia **openUI** puede actuar:

- ConsoleForm
- Un consoleField o uno de los siguientes elementos:
  - Una lista de consoleFields, cada uno separado del siguiente por una coma.
  - Un dictionary declarado en un consoleForm y que haga referencia a un conjunto de consoleFields de dicho consoleForm
  - Un arrayDictionary declarado en un consoleForm y que hace referencia a un conjunto de matrices de consoleField de dicho consoleForm
- Menu
- Prompt
- Window

*BindClause*

La lista de variables primitivas, registros o matrices que están enlazadas a las variables de ConsoleUI. Las características de las variables de enlace dependen de las características de la variable de consoleUI en la que actúe la sentencia **openUI**:

- Para un consoleField, puede especificar una variable primitiva.

- Para un arrayDictionary en pantalla, puede especificar una matriz de registros, un elemento por fila en el arrayDictionary; y, si cada fila del arrayDictionary representa un solo valor, puede especificar una matriz de variables primitivas.
- Para un dictionary o una lista de consoleFields, puede especificar una lista de variables primitivas. Como alternativa, puede especificar una matriz de registros, cada uno de cuyos elementos contenga una serie de campos enlazados a los consoleFields. Esta alternativa es equivalente a enlazar una matriz dinámica con un arrayDictionary en pantalla que sólo tenga una fila; puede añadir, insertar o suprimir un registro para cambiar la matriz dinámica, y en cualquier caso sólo se visualizará un registro simultáneamente.
- Para una solicitud, puede especificar un campo primitivo que reciba la respuesta del usuario.

Para obtener información detallada sobre enlaces, consulte el apartado de *OnEventBlock* (más adelante), así como el apartado *Componentes de ConsoleUI y variables relacionadas*.

### *OnEventBlock*

Un *bloque de evento* es una estructura de programación que incluye de 0 a muchos *manejadores de eventos*, cada uno de los cuales contiene el código que se ha escrito para responder a un determinado evento. Un manejador de eventos empieza con una cabecera *OnEvent*:

```
OnEvent(claseEvento: calificadorEvento)
```

#### *claseEvento*

Uno de varios eventos. Los valores válidos se describen en el apartado “Tipos de eventos” en la página 627.

#### *calificadorEvento*

Datos que definen más el evento. Estos datos podrían ser el ConsoleField especificado o la pulsación de tecla realizada.

Las sentencias EGL que responden a un determinado evento están entre la cabecera *OnEvent* y la siguiente cabecera *OnEvent* (si existe), como se muestra más adelante en un ejemplo:

El usuario interactúa continuamente con el programa, y éste ejecuta un manejador de eventos cuando se produce el evento asociado con el manejador de eventos. Sin embargo, si la finalidad de la sentencia **openUI** es visualizar una solicitud, la interacción del programa de usuario es menos parecida a bucle:

1. Un manejador de eventos (potencialmente uno de varios) captura una pulsación de usuario y responde
2. La sentencia **openUI** finaliza

Ningún bloque de evento está disponible para una ventana.

Considere el siguiente ejemplo para guiar la interacción entre el usuario y un ConsoleForm:

```
openUI {bindingByName=yes}
 activeForm
 bind firstName, lastName, ID
 OnEvent(AFTER_FIELD:"ID")
 if (employeeID == 700)
```

```

 firstName = "Angela";
 lastName = "Smith";
 end
end

```

Este código actúa de la manera siguiente:

- Abre el *ConsoleForm activo* (que es el consoleForm que se ha visualizado más recientemente en la ventana activa);
- Enlaza un conjunto de variables primitivas a cada uno de los ConsoleFields; y
- Especifica que después de que el usuario escriba un valor en *employeeID* y abandone dicho ConsoleField, EGL coloca series en otras dos variables.

Considere estos detalles relativos al ejemplo anterior:

- El cursor se inicia en el primero de los consoleFields de la lista; pero debe iniciarse en el consoleField ID a fin de que el manejador de eventos no borre la entrada del usuario de los demás consoleFields.
- El manejador de eventos actualiza las variables enlazadas a los consoleFields *firstName* y *lastName*, pero no provoca la visualización de esos valores hasta que el cursor entra en esos campos. Puede que desee visualizar antes esos valores.

Puede finalizar una sentencia **openUI** emitiendo una sentencia **exit** con el formato **exit openUI**.

## Tipos de eventos

ConsoleUI da soporte a los siguientes eventos:

### BEFORE\_OPENUI

El entorno de ejecución EGL empieza a ejecutar la sentencia **OpenUI**. Este evento está disponible para todas las variables ConsoleUI excepto las basadas en Window.

### AFTER\_OPENUI

El entorno de ejecución EGL está a punto de detener la ejecución de la sentencia **OpenUI**. Este evento está disponible para todas las variables ConsoleUI excepto las basadas en Window.

### ON\_KEY:(ListaDeSeries)

El usuario ha pulsado una tecla específica, indicado por una serie, como por ejemplo "ESC", "F2" o "CONTROL\_W". Puede identificar varias teclas separando una serie de la siguiente, como en este ejemplo:

```
ON_KEY:("a", "ESC")
```

Este evento está disponible para todas las variables ConsoleUI excepto las basadas en Window.

### BEFORE\_FIELD:(ListaDeSeries)

El usuario ha movido el cursor al ConsoleField especificado, como se indica mediante una serie que coincide con el valor del campo Nombre de ConsoleField. Puede identificar varios consoleField en el mismo consoleForm separando una serie de la siguiente, como se indica en este ejemplo:

```
BEFORE_FIELD:("field01", "field02")
```

### AFTER\_FIELD:(ListaDeSeries)

El usuario ha movido el cursor fuera del ConsoleField especificado, como se indica mediante una serie que coincide con el valor del campo Nombre de ConsoleField. Puede identificar varios consoleField en el mismo consoleForm separando una serie de la siguiente, como se indica en este ejemplo:

```
AFTER_FIELD:("field01", "field02")
```

#### **BEFORE\_DELETE**

En relación con un `arrayDictionary` en pantalla, el usuario ha pulsado la tecla especificada en `ConsoleLib.key_deleteLine`, pero el entorno de ejecución EGL todavía no ha suprimido la fila. El programa puede invocar a `consoleLib.cancelDelete` para evitar suprimir la fila.

#### **BEFORE\_INSERT**

En relación con un `arrayDictionary` en pantalla, el usuario ha pulsado la tecla especificada en `ConsoleLib.key_insertLine`, pero el entorno de ejecución EGL todavía no ha insertado una fila. El programa puede invocar a `consoleLib.cancelInsert` para evitar insertar la fila.

#### **BEFORE\_ROW**

En relación con un `arrayDictionary` en pantalla, el usuario ha movido el cursor a una fila.

#### **AFTER\_DELETE**

En relación con un `arrayDictionary` en pantalla, el usuario ha pulsado la tecla especificada en `ConsoleLib.key_deleteLine`, y el entorno de ejecución EGL ha suprimido una fila.

#### **AFTER\_INSERT**

En relación con un `arrayDictionary` en pantalla, el usuario ha pulsado la tecla especificada en `ConsoleLib.key_insertLine`; el entorno de ejecución EGL ha insertado una fila; y el cursor abandona la fila insertada.

El usuario puede editar la fila antes de comprometer los cambios a una base de datos, como ocurre generalmente en el manejador `AFTER_INSERT`.

#### **AFTER\_ROW**

El usuario ha movido el cursor fuera de una fila en un `arrayDictionary` en pantalla.

#### **MENU\_ACTION:(ListaDeSeries)**

El usuario ha seleccionado un `menuItem`, como se indica mediante una serie que coincide con el valor del campo `Nombre` de `menuItem`. Puede identificar varios `menuItems` separando una serie de la siguiente, como en este ejemplo:

```
MENU_ACTION:("item01", "item02")
```

### **isConstruct**

Cuando la propiedad `isConstruct = yes`, el texto colocado en la variable enlazada al mandato `openUI` tiene un formato especial, como se muestra en el siguiente ejemplo:

1. Una sentencia `openUI` actúa sobre un `ConsoleForm` de tres `ConsoleFields` (*employee*, *age* y *city*), y cada campo está asociado a una columna de tabla SQL del mismo nombre.

Un `consoleField` se asocia a una columna de tabla SQL estableciendo la propiedad `SQLColumnName` de `ConsoleField`, y el usuario debe establecer la propiedad `dataType` de `consoleField`, como se indica en el apartado *Propiedades y campos de ConsoleField*.

2. El usuario actúa de la manera siguiente:
  - Deja en blanco el campo *employee*
  - Escribe lo siguiente en el campo *age*:  
> 25
  - Escribe lo siguiente en el campo *city*:  
= 'Sarasota'

3. Cuando el usuario abandona la variable en pantalla sobre la que actúa la sentencia openUI, la variable enlazada recibe el siguiente contenido:

AGE > 28 AND CITY = 'Sarasota'

Como se muestra en el ejemplo, EGL coloca el operador AND entre cada cláusula que proporciona el usuario.

La tabla siguiente muestra la entrada de usuario válida y la cláusula resultante. La expresión *tipos SQL simples* hace referencia a tipos SQL que no son estructurados ni de tipo LOB.

Símbolo	Definición	Tipos de datos soportados	Ejemplo	Cláusula resultante (para una columna de caracteres llamada C)	Cláusula resultante (para una columna numérica llamada C)
=	Igual a	Tipos SQL simples	=x, ==x	C = 'x'	C = x
>	Mayor que	Tipos SQL simples	>x	C > 'x'	C > x
<	Menor que	Tipos SQL simples	<x	C < 'x'	C < x
>=	No menor que	Tipos SQL simples	>=x	C >= 'x'	C >= x
<=	No mayor que	Tipos SQL simples	<=x	C <= 'x'	C <= x
<> o !=	No igual a	Tipos SQL simples	<>x o !=x	C != 'x'	C != x
..	Rango	Tipos SQL simples	x.y o x..y	C BETWEEN 'x' AND 'y'	C BETWEEN x AND y
*	Comodín para String (como se describe en la tabla siguiente)	CHAR	*x o x* o *x*	C MATCHES '*x'	no aplicable
?	Comodín de un solo carácter (como se describe en la tabla siguiente)	CHAR	?x, x?, ?x?, x??	C MATCHES '?x'	no aplicable
	O lógico	Tipos SQL simples	x y	C IN ('x', 'y')	C IN (x, y)

El signo igual (=) puede significar IS NULL; y el signo no igual (!= o <>) puede significar IS NOT NULL.

Una cláusula MATCHES es el resultado de la especificación por parte del usuario de uno de los caracteres comodín descritos en la tabla siguiente.

Símbolo	Efecto
---------	--------

*	Coincide con cero o más caracteres.
?	Coincide con un solo carácter cualquiera.
[ ]	Coincide con cualquier carácter encerrado entre corchetes.
- (guión)	Cuando se utiliza entre caracteres que están encerrados entre corchetes, un guión coincide con cualquier carácter del rango entre los dos caracteres, ambos inclusive. Por ejemplo, [a-z] coincide con cualquier letra en minúsculas o carácter especial dentro del rango en minúsculas.
^	Cuando se utiliza entre corchetes, un signo de intercalación inicial coincide con cualquier carácter no incluido dentro de los corchetes. Por ejemplo, [^abc] coincide con cualquier carácter excepto a, b, c.
\	Es el carácter de escape; el siguiente carácter es un literal. Permite incluir cualquiera de los caracteres comodín en la serie sin tener el efecto de comodín.
Cualquier otro carácter fuera de los corchetes	Debe coincidir exactamente.

### Conceptos relacionados

“Interfaz de usuario de consola” en la página 177

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

“Componentes de ConsoleUI y variables relacionadas” en la página 180

“Opciones de pantalla ConsoleUI para UNIX” en la página 184

“Programa de UI de texto en formato fuente EGL” en la página 731

### Tareas relacionadas

“Crear una interfaz con consoleUI” en la página 178

## prepare

La sentencia EGL **prepare** especifica una sentencia SQL PREPARE, que opcionalmente incluye detalles que sólo se conocen durante la ejecución. La sentencia SQL preparada se ejecuta mediante la ejecución de una sentencia EGL **execute** o (si la sentencia SQL devuelve un conjunto de resultados) mediante la ejecución de una sentencia EGL **open** o **get**.



#### *IdSentenciaPreparada*

Identificador que relaciona la sentencia **prepare** con la sentencia **execute**, **open** o **get**.

#### *expresiónSerie*

Una *expresión de serie* que contiene una sentencia SQL válida.

#### *nombre de registro SQL*

Nombre de un registro SQL. La especificación de este nombre ofrece dos ventajas:

- El editor EGL suministra un diálogo para que permite derivar una sentencia SQL basada en las especificaciones del usuario
- Una vez ejecutada la sentencia **prepare**, puede probar el nombre de registro con un valor de error de E/S para determinar si la sentencia ha sido satisfactoria, como en el ejemplo siguiente:

```

try
 prepare prep01 from
 "insert into " + aTableName +
 "(empnum, empname) " +
 "value ?, ?"
 for empRecord;

onException
 if empRecord is unique
 myErrorHandler(8);
 else
 myErrorHandler(12);
 end
end

```

A continuación se ofrece otro ejemplo:

```

myString =
 "insert into myTable " + "(empnum, empname) " +
 "value ?, ?";

try
 prepare myStatement
 from myString;
onException
 myErrorHandler(12); // sale del programa
end

try
 execute myStatement
 using :myRecord.empnum,
 :myRecord.empname;
onException
 myErrorHandler(15);
end

```

Como se muestra en los ejemplos anteriores, el desarrollador puede utilizar un signo de interrogación (?) cuando debe aparecer una variable de lenguaje principal. A continuación, el nombre de la variable de lenguaje principal utilizada durante la ejecución se coloca en la cláusula using de la sentencia **execute**, **open** o **get** que ejecuta la sentencia preparada.

Una sentencia **prepare** que actúa en una fila de un conjunto de resultados puede incluir una frase en el formato *where current of resultSetIdentifier*. Esta técnica sólo es válida en la siguiente situación:

- La frase está codificada en un literal; y
- El conjunto de resultados está abierto cuando se ejecuta la sentencia **prepare**.

A continuación se ofrece un ejemplo:

```

prepare prep02 from
 "update myTable " +
 "set empname = ?, empphone = ? where current of x1" ;

execute prep02 using empname, empphone ;
freeSQL prep02;

```

Para obtener un ejemplo de cómo afectan los paréntesis a la utilización de un signo más (+), consulte la sección *Expresiones*.

#### Conceptos relacionados

“Referencias a componentes” en la página 21  
“Tipos de registros y propiedades” en la página 135  
“Soporte de SQL” en la página 229

#### Consulta relacionada

“add” en la página 561  
“close” en la página 568  
“delete” en la página 571  
“Manejo de excepciones” en la página 94  
“execute” en la página 574  
“Expresiones” en la página 495  
“freeSQL” en la página 584  
“get” en la página 585  
“get next” en la página 597  
“get previous” en la página 602  
“Valores de error de E/S” en la página 536  
“Sentencias EGL” en la página 88  
“open” en la página 616  
“replace”  
“Propiedades de elementos SQL” en la página 67  
“Expresiones de texto” en la página 505  
“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

## print

La sentencia EGL **print** añade un formulario de impresión a un almacenamiento intermedio de tiempo de ejecución, como se describe en el apartado *Formularios de impresión*.

► print nombFormImpr ; —————▶

#### nombreFormularioImpresión

Nombre del formulario de impresión que es visible para el programa. Para obtener detalles acerca de la visibilidad, consulte el apartado *Referencias a componentes*.

#### Conceptos relacionados

“Formularios de impresión” en la página 156  
“Referencias a componentes” en la página 21

## replace

La sentencia EGL **replace** coloca un registro cambiado en un archivo o base de datos.



*nombre de registro*

Nombre del objeto de E/S; un registro indexado, relativo o SQL.

**with #sql{ sentencia SQL }**

Una sentencia SQL UPDATE explícita. No deje espacios entre #sql y el corchete de apertura.

**from IDconjuntoResultados**

ID que conecta la sentencia **replace** con una sentencia **get** o **open** ejecutada antes en el mismo programa. Para obtener detalles, consulte el apartado *resultSetID*.

El ejemplo siguiente muestra cómo leer y sustituir un registro de archivo:

```

emp.empnum = 1; // establece la clave en el registro emp

try
 get emp forUpdate;
onException
 myErrorHandler(8); // sale del programa
end

emp.empname = emp.empname + " Smith";

try
 replace emp;
onException
 myErrorHandler(12);
end

```

Los detalles de la sentencia **replace** dependen del tipo de registro. Para obtener detalles acerca del proceso SQL, consulte el tema *Registro SQL*.

### Registro indexado o relativo

Si desea sustituir un registro indexado o relativo, debe emitir una sentencia **get** para el registro con la opción *forUpdate*, y a continuación emitir la sentencia **replace** sin que intervenga ninguna operación de E/S en el mismo archivo.

Después de invocar la sentencia **replace**, el resultado de la próxima operación de E/S en el mismo archivo será el siguiente:

- Si la próxima operación de E/S es una sentencia **replace** en el mismo registro EGL, se cambia el registro del archivo
- Si la próxima operación de E/S es una sentencia **delete** en el mismo registro EGL, el registro del archivo se marca para supresión
- Si la próxima operación de E/S es una sentencia **get** en un registro del mismo archivo e incluye la opción *forUpdate*, una sentencia **replace** o **delete** subsiguiente es válida en el registro de archivo de lectura recién creado
- Si la próxima operación de E/S es una sentencia **get** en el mismo registro EGL (sin la opción *forUpdate*) o es una sentencia **close** en el mismo archivo, el registro de archivo se libera sin cambios

Encontrará los detalles sobre la opción `forUpdate` en `get`.

## Registro SQL

En el caso de procesos SQL, la sentencia EGL **replace** provoca una sentencia SQL UPDATE en el código generado.

Debe recuperar una fila para la sustitución subsiguiente, de cualquiera de estas dos formas:

- Emita una sentencia **get** (con la opción `forUpdate`) para recuperar la fila; o
- Emita una sentencia **open** para seleccionar un conjunto de filas y, a continuación, invoque una sentencia **get next** para recuperar la fila en cuestión.

**Condiciones de error:** Las condiciones siguientes se encuentran entre las que no son válidas al utilizar una sentencia **replace**:

- Se especifica una sentencia SQL de un tipo que no es UPDATE
- Se especifica alguna, pero no todas las cláusulas de una sentencia SQL UPDATE
- No se especifica un valor *resultSetID* cuando es necesario; para obtener detalles, consulte el apartado *resultSetID*
- Se especifica (o se acepta) una sentencia UPDATE que tiene una de estas características:
  - Actualiza varias tablas
  - Está asociada con una columna que no existe o que es incompatible con la variable de lenguaje principal relacionada
- Se utiliza un registro SQL como objeto de E/S, y todos los elementos de registro son de sólo lectura

La situación siguiente también provoca un error:

- Personaliza una sentencia EGL **get** con la opción `forUpdate`, pero no puede indicar que una columna de tabla SQL determinada está disponible para actualización; y
- La sentencia **replace** que está relacionada con la sentencia **get** intenta revisar la columna.

Puede resolver la discrepancia anterior de cualquiera de estas formas:

- Al personalizar la sentencia EGL **get**, incluya el nombre de columna en la sentencia SQL SELECT, cláusula FOR UPDATE OF; o
- Al personalizar la sentencia EGL **replace**, elimine la referencia a la columna en la sentencia SQL UPDATE, cláusula SET; o
- Acepte los valores por omisión para las sentencias **get** y **replace**.

**Sentencia SQL implícita:** Por omisión, el resultado de una sentencia **replace** que escribe un registro SQL es el siguiente:

- Como resultado de la asociación de elementos de registro y columnas de tabla SQL en la declaración de registro, el código generado copia los datos de cada elemento de registro en la columna de tabla SQL relacionada.
- Si ha definido un elemento de registro como de sólo lectura, el valor de la columna que corresponde a ese elemento de registro no resulta afectado

La sentencia SQL tiene estas características por omisión:

- La sentencia SQL UPDATE no incluye los elementos de registro de sólo lectura
- La sentencia SQL UPDATE de un registro determinado es similar a la sentencia siguiente:

```

UPDATE nombreTabla
SET columna01 = :elementoRegistro01,
 columna02 = :elementoRegistro02,
 .
 .
 .
 columnaNn = :elementoRegistroNn WHERE CURRENT OF cursor

```

### Conceptos relacionados

"Tipos de registros y propiedades" en la página 135  
 "Referencias a componentes" en la página 21  
 "resultSetID" en la página 743  
 "Unidad de ejecución" en la página 742  
 "Soporte de SQL" en la página 229

### Tareas relacionadas

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

### Consulta relacionada

"add" en la página 561  
 "close" en la página 568  
 "delete" en la página 571  
 "Sentencias EGL" en la página 88  
 "Manejo de excepciones" en la página 94  
 "execute" en la página 574  
 "get" en la página 585  
 "get next" en la página 597  
 "get previous" en la página 602  
 "Valores de error de E/S" en la página 536  
 "open" en la página 616  
 "prepare" en la página 630  
 "Propiedades de elementos SQL" en la página 67  
 "terminalID" en la página 938

## return

La sentencia EGL **return** sale de una función y, opcionalmente, devuelve un valor a la función invocante.



### valorRetorno

Un elemento, literal o constante que es compatible con la especificación **returns** de la declaración de función EGL.

Aunque un elemento debe corresponder de todas formas a la especificación **returns**, las normas de literales y constantes son las siguientes:

- Un literal o constante numérica sólo puede devolverse si el tipo primitivo de la especificación **returns** es un tipo numérico
- Un literal o constante que incluya sólo caracteres de un solo byte sólo puede devolverse si el tipo primitivo de la especificación **returns** es CHAR o MBCHAR

- Un literal o constante que incluya sólo caracteres de doble byte sólo puede devolverse si el tipo primitivo de la especificación **returns** es DBCHAR
- Un literal o constante que incluya una combinación de caracteres de un solo byte y de doble byte sólo puede devolverse si el tipo primitivo de la especificación **returns** es MBCHAR
- Un literal o constante no puede devolverse si el tipo primitivo de la especificación **returns** es HEX

Una función que incluya una especificación **returns** debe terminar con una sentencia **return** que incluya un valor. Una función que no tenga una especificación **returns** puede terminar con una sentencia **return**, que no debe incluir un valor.

La sentencia **return** otorga el control a la primera sentencia que sigue a la invocación de la función, aunque la sentencia se encuentre en una cláusula **OnException** de un bloque **try**.

## set

Las secciones que siguen describen el efecto de una sentencia EGL **set**:

- “Efecto sobre un registro (o un registro fijo) como un todo”
- “Efecto sobre la totalidad de un formulario” en la página 638
- “Efecto sobre un campo en cualquier contexto” en la página 639
- “Efecto sobre un campo de un formulario de texto” en la página 640

### Efecto sobre un registro (o un registro fijo) como un todo

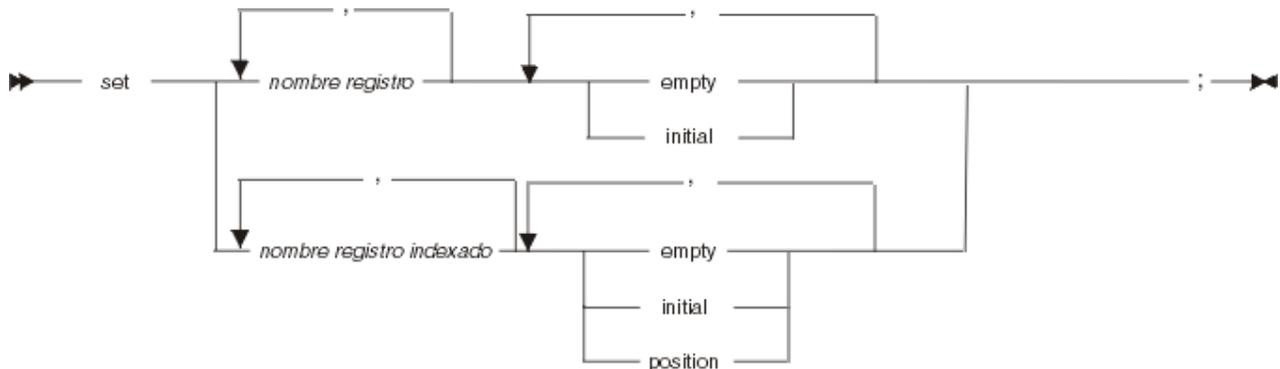
La tabla siguiente describe las sentencias **set** que afectan a un registro como un todo, aun registro fijo como un todo o a una matriz de cualquiera de los dos.

Formato de sentencia set	Efecto
set record empty	<p>Vacía cada uno de los campos elementales. Para un registro, se vacía cada registro subordinado, como está cada subordinado de esos subordinados, etc. Para un registro fijo (que puede estar en un registro), los campos elementales están en el nivel más bajo de la estructura fija.</p> <p>El efecto sobre cada campo elemental depende del tipo primitivo de ese campo:</p> <ul style="list-style-type: none"> <li>• Para un campo de tipo ANY, la sentencia <b>set</b> inicializa el campo de acuerdo con el tipo actual del campo y si el campo es de tipo ANY y no tiene otro tipo, la sentencia <b>set</b> no surte efecto</li> <li>• Para obtener detalles sobre campos de otros tipos, consulte la sección <i>Inicialización de datos</i></li> </ul>

Formato de sentencia set	Efecto
set record initial	<p>Restablece los valores de campo en los especificados por la propiedad <b>value</b> durante el desarrollo, tal como es posible para un registro o registro fijo declarado en un pageHandler o formulario. Un valor establecido por asignación no se reintegra nunca.</p> <p>Si la propiedad <b>value</b> no tiene ningún valor o si el registro no está en un pageHandler o formulario, el efecto de <i>set record initial</i> es el mismo que el efecto de <i>set record empty</i> con una excepción: para un campo de tipo ANY, la sentencia <b>set</b> elimina cualquier especificación de tipo distinto a ANY.</p>
set record position	<p>Establece la posición en el archivo VSAM asociado a un registro fijo de tipo indexedRecord, tal como se describe posteriormente.</p> <p>Este formato de sentencia set no está disponible para una matriz.</p>

Puede combinar formatos de sentencia insertando una coma para separar las opciones. Para un recuerdo dado, las opciones entran en vigor por el orden en el que aparecen en la sentencia **set**. Además, también puede especificar varios registros insertando una coma para separar uno de otro.

El diagrama de sintaxis es el siguiente:



*nombre de registro*

Nombre de un registro o de un registro fijo de cualquier tipo. Puede especificar una matriz.

*nombre de registro indexado*

Nombre de un registro fijo de tipo indexedRecord. Puede especificar una matriz solo si no incluye *set record position*.

**empty**

Como se ha descrito en la tabla anterior.

**initial**

Como se ha descrito en la tabla anterior.

### position

Establece una posición de archivo basada en el *valor de set*, que es el valor de clave de un registro indexado. El efecto global depende de la siguiente operación de entrada o salida realizada por el código en el mismo registro indexado:

- Si la siguiente operación es una sentencia EGL **get next**, dicha sentencia lee el primer registro de archivo cuyo valor de clave es igual o superior al valor de set. Si no existe tal registro, el resultado de la sentencia **get next** es **endOfFile**.
- Si la operación posterior a una sentencia *set record position* es una sentencia EGL **get previous**, dicha sentencia lee el primer registro de archivo cuyo valor de clave es igual o inferior al valor de set. Si no existe tal registro, el resultado de la sentencia **get previous** es **endOfFile**.
- Cualquier otra operación posterior a *set record position* restablece la posición de archivo, y la sentencia *set record position* no tiene ningún efecto.

Si el valor de set está relleno con caracteres FF hexadecimales, se realiza lo siguiente:

- La sentencia *set record position* establece una posición de archivo posterior al último registro del archivo
- Si la próxima operación es una sentencia **get previous**, se recupera el último registro del archivo

### Efecto sobre la totalidad de un formulario

La tabla siguiente describe las sentencias **set** que afectan a la totalidad de un formulario.

Formato de sentencia set	Efecto
set form alarm	Sólo para formularios de texto; hace sonar una alarma la próxima vez que una sentencia <b>converse</b> presenta el formulario.
set form empty	Vacía el valor de cada campo en el formulario, borrando el contenido. El efecto sobre un campo determinado depende del tipo primitivo: <ul style="list-style-type: none"><li>• Para un campo de tipo ANY, la sentencia <b>set</b> inicializa el campo de acuerdo con el tipo actual del campo y si el campo es de tipo ANY y no tiene otro tipo, la sentencia <b>set</b> no surte efecto</li><li>• Para obtener detalles sobre campos de otros tipos, consulte la sección <i>Inicialización de datos</i></li></ul>
set form initial	Restablece cada campo del formulario en su estado definido originariamente, según lo indicado en la declaración de formulario. Los cambios efectuados por el programa se cancelan. Para un campo de tipo ANY, la sentencia <b>set</b> elimina cualquier especificación cuyo tipo sea distinto a ANY.

Formato de sentencia set	Efecto
set form initialAttributes	Restablece cada campo del formulario en su estado definido originariamente, según lo indicado en la declaración de formulario. El contenido del campo no se ve afectado, ni (en el caso de un campo de tipo ANY) es el tipo afectado.

Puede combinar formatos de sentencia insertando una coma para separar opciones como **empty** y **alarm**. Además, también puede especificar varios formularios insertando una coma para separar un formulario del siguiente.

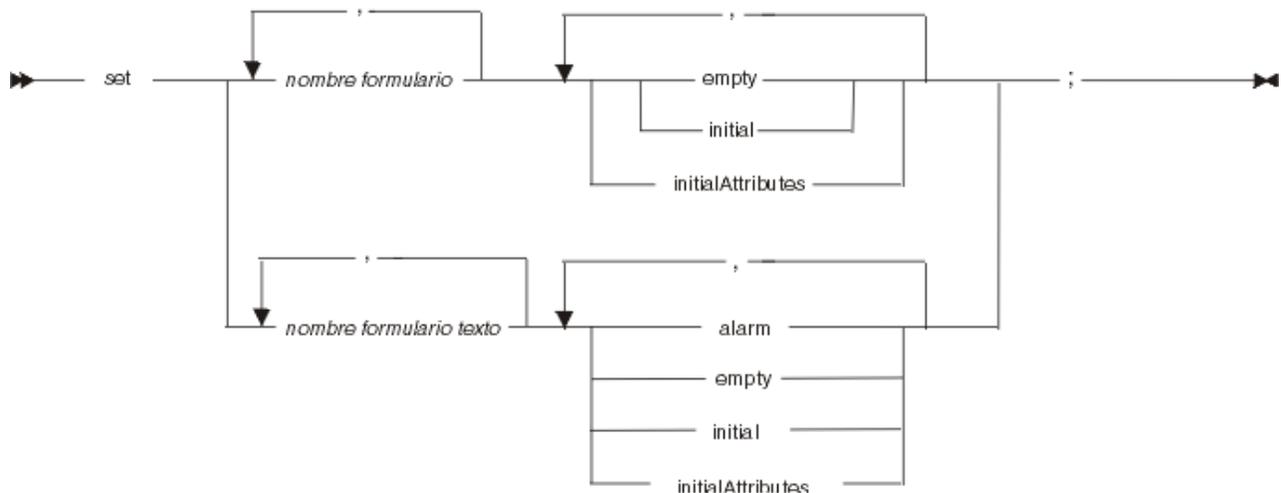
Puede elegir uno o ninguno de los siguientes formatos:

- *set form empty*
- *set form initial*

Puede elegir uno, ambos o ninguno de los siguientes formatos:

- *set form alarm* (disponible sólo para formularios de texto)
- *set form initialAttributes*

El diagrama de sintaxis es el siguiente:



*nombre de formulario*

Nombre de un formulario de tipo *texto* o *impresión*, tal como se describe en el apartado *Componente de formulario*.

*nombre de formulario de texto*

Nombre de un formulario de tipo *texto*, como se describe en el apartado *Componente de formulario*.

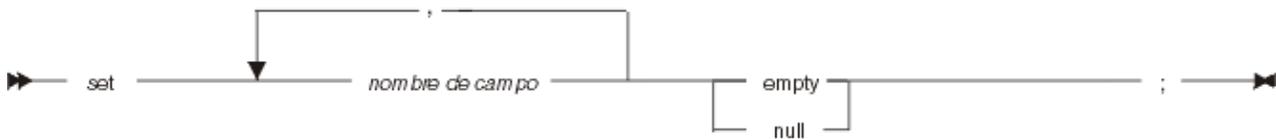
Las opciones son las descritas en la tabla anterior.

### **Efecto sobre un campo en cualquier contexto**

La tabla siguiente describe el formato de la sentencia **set** que afecta a un campo en cualquier contexto.

Formato de sentencia set	Efecto
set field empty	<p>Vacía el campo o (para un campo fijo con una subestructura) vacía cada campo elemental, subordinado.</p> <p>El efecto depende del tipo primitivo de un campo:</p> <ul style="list-style-type: none"> <li>• Para un campo de tipo ANY, la sentencia <b>set</b> inicializa el campo de acuerdo con el tipo actual del campo y si el campo es de tipo ANY y no tiene otro tipo, la sentencia <b>set</b> no surte efecto</li> <li>• Para obtener detalles sobre campos de otros tipos, consulte la sección <i>Inicialización de datos</i></li> </ul>
set field null	<p>Anula el campo, si ello es válido. Para conocer detalles acerca de cuándo es válida la operación, consulte la sección <i>itemsNullable</i>. Para conocer detalles de proceso nulo en registros SQL, consulte la sección <i>Propiedades del elemento SQL</i>.</p>

El diagrama de sintaxis es el siguiente:



*nombre de campo*  
Nombre del campo.

Puede seleccionar una u otra opción y cada una se describe en la tabla anterior.

### Efecto sobre un campo de un formulario de texto

La tabla siguiente describe las sentencias **set** que afectan a un campo o a una matriz de campos de un formulario de texto. Una sentencia **set** determinada puede combinar opciones solamente mediante un conjunto de métodos determinado, como se describe más adelante.

**Nota:** Muchas de las acciones descritas dependen del dispositivo en el que se visualice el formulario de texto. Se recomienda probar la salida de cada uno de los dispositivos soportados.

Formato de sentencia set	Efecto
set field blink	Provoca que el texto parpadee repetidamente. Esta opción solo está disponible en programas COBOL.
set field bold	Hace que el texto aparezca en negrita.

Formato de sentencia set	Efecto
set field cursor	<p>Sitúa el cursor en el campo especificado.</p> <p>Si el campo identifica una matriz y no tiene valor de apariciones, el cursor se sitúa en el primer elemento de matriz por omisión.</p> <p>Si el programa ejecuta varias sentencias en el formato <i>set field cursor</i>, la última de ellas estará en vigor cuando se ejecute la sentencia <b>converse</b>.</p>
set field defaultColor	<p>Establece la propiedad <b>color</b> específica del campo en <i>defaultColor</i>, lo que significa que otras condiciones determinan el color visualizado. Para obtener detalles, consulte el apartado <i>Propiedades de presentación de campos</i>.</p>
set field dim	<p>Hace que el campo aparezca con menos intensidad de la normal. Utilice este efecto para quitar énfasis al contenido de un campo.</p>
set field empty	<p>Inicializa el valor del campo, borrando el contenido. El efecto sobre un campo determinado depende del tipo primitivo, como se describe en el apartado <i>Inicialización de datos</i>.</p>

Formato de sentencia set	Efecto
set field full	<p>Establece un campo vacío, en blanco o nulo en una serie de caracteres idénticos antes de que se presente el formulario:</p> <ul style="list-style-type: none"> <li>• El carácter es un asterisco (*) si la propiedad de campo <b>fillCharacter</b> tiene el siguiente valor (que es también el valor por omisión para <b>fillCharacter</b>): <ul style="list-style-type: none"> <li>– 0 para campos de tipo HEX</li> <li>– espacio para campos de tipo numérico</li> <li>– serie vacía para otros campos</li> </ul> </li> <li>• Si <b>fillCharacter</b> no se establece como se ha descrito, el carácter será idéntico al valor de <b>fillCharacter</b>.</li> </ul> <p>Los caracteres del formulario sólo se devuelven al programa si el código de datos modificados del campo se establece según lo descrito en el apartado <i>Código y propiedad de datos modificados</i>. Un usuario que cambie el campo debe eliminar todos los caracteres que se encuentren en el campo para evitar que vuelvan al programa.</p> <p>La utilización de <i>set field full</i> sólo tiene efecto si el grupo de formularios se genera con la opción <i>setFormItemFull</i> del descriptor de construcción.</p> <p>Un campo de tipo MBCHAR se considera vacío si sólo contiene espacios de un solo byte. En relación a este tipo de campos, <i>set field full</i> asigna una serie de caracteres de un solo byte.</p>
set field initial	<p>Vuelve a establecer el campo en el estado definido originariamente, independientemente de los cambios efectuados por el programa</p>
set field initialAttributes	<p>Vuelve a establecer el campo en el estado definido originariamente, sin utilizar la propiedad <b>value</b> (que especifica el contenido actual del campo)</p>
set field invisible	<p>Hace invisible el texto del campo</p>
set field masked	<p>Adecuado para los archivos de contraseña. Si el formulario de texto lo presenta un programa Java, se visualiza un asterisco en lugar de cualquier carácter distinto de un espacio en blanco que teclee el usuario en un campo de entrada.</p>
set field modified	<p>Establece el código de datos modificados según lo descrito en el apartado <i>Código y propiedad de datos modificados</i>.</p>
set field noHighlight	<p>Elimina los efectos especiales de parpadeo, inversión y subrayado.</p>

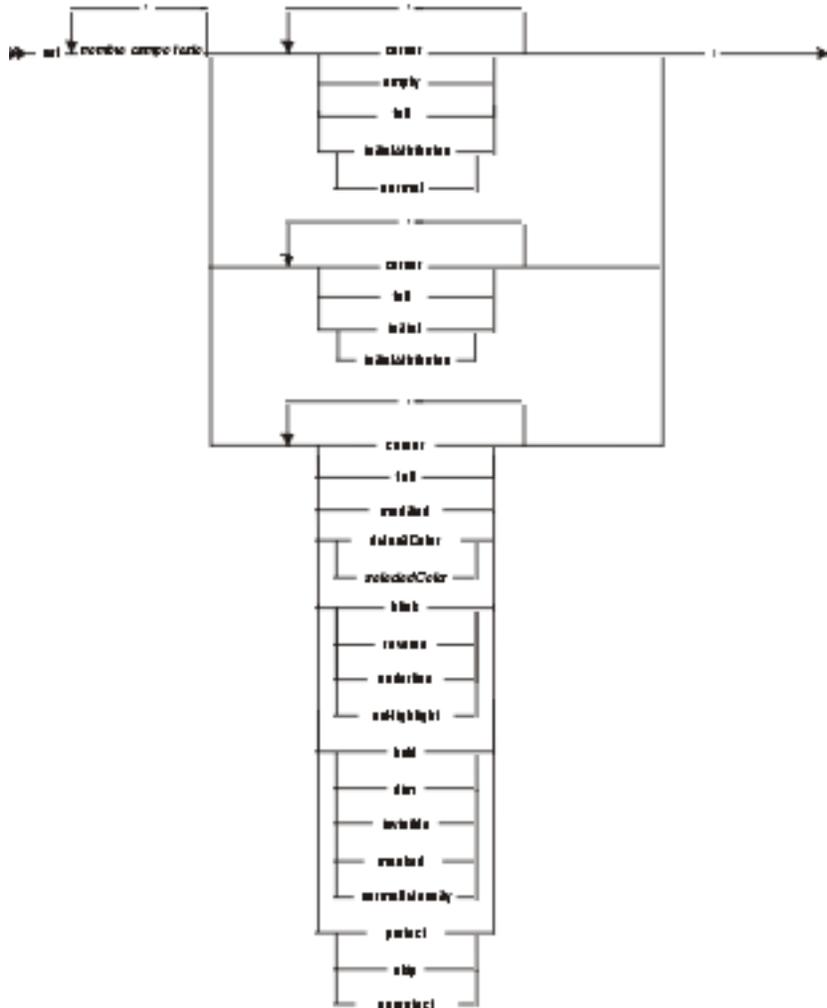
Formato de sentencia set	Efecto
set field normal	Vuelve a establecer los campos según lo descrito en relación a los siguientes formatos: <ul style="list-style-type: none"> <li>• Set field normalIntensity</li> <li>• Set field unmodified</li> <li>• Set field unprotected</li> </ul> Para obtener detalles, consulte la tabla siguiente.
set field normalIntensity	Establece el campo como visible, sin negrita.
set field protect	Establece el campo de forma que el usuario no pueda sobrescribir el valor en él. Consulte también el formato <i>set field skip</i> .
set field reverse	Invierte los colores del texto y del fondo, de forma que (por ejemplo), si la pantalla tiene un fondo oscuro y el texto claro, el fondo pasa a ser claro y el texto pasa a ser oscuro.
set field <i>colorSeleccionado</i>	Establece la propiedad <b>color</b> en el valor especificado. Los valores válidos para <i>colorSeleccionado</i> son los siguientes: <ul style="list-style-type: none"> <li>• black</li> <li>• blue</li> <li>• green</li> <li>• pink</li> <li>• red</li> <li>• turquoise</li> <li>• white</li> <li>• yellow</li> </ul>
set field skip	Establece el campo de forma que el usuario no pueda sobrescribir el valor en él. Además, el cursor pasa por alto el campo en cualquiera de estos casos: <ul style="list-style-type: none"> <li>• El usuario trabaja en el campo anterior por orden de tabulación y pulsa el <b>tabulador</b> o rellena ese campo anterior con contenido; o</li> <li>• El usuario trabaja en el campo siguiente por orden de tabulación y pulsa <b>Mayúsculas Tabulador</b>.</li> </ul>
set field underline	Coloca un subrayado en la parte inferior del campo.
set field unprotect	Establece el campo de forma que el usuario pueda sobrescribir el valor en él.

Puede combinar formatos de sentencia insertando una coma para separar opciones como **cursor** y **full**, de cualquiera de estas formas:

1. Puede construir una sentencia **set** del siguiente modo:
  - Elija uno o ninguno de estos formatos de atributo de campo:
    - *set field initialAttributes*
    - *set field normal*

- Elija cualquier número de los siguientes formatos:
  - *set field cursor*
  - *set field empty*
  - *set field full*
- 2. En segundo lugar, puede construir una sentencia **set** a partir de cualquier número de formatos de texto:
  - *set field cursor*
  - *set field full*
  - *set field initial* o *set field initialAttributes*
- 3. Por último, puede construir una sentencia **set** del siguiente modo:
  - Elija cualquier número de los siguientes formatos:
    - *set field cursor*
    - *set field full*
    - *set field modified*
  - Elija uno o ninguno de los formatos de color:
    - *set field defaultColor*
    - *set field selectedColor*
  - Elija uno o ninguno de los formatos de resaltado:
    - *set field blink*
    - *set field reverse*
    - *set field underline*
    - *set field noHighlight*
  - Elija uno o ninguno de los formatos de intensidad:
    - *set field bold*
    - *set field dim*
    - *set field invisible*
    - *set field masked*
    - *set field normalIntensity*
  - Elija uno o ninguno de los formatos de protección:
    - *set field protect*
    - *set field skip*
    - *set field unprotect*

El diagrama de sintaxis es el siguiente:



*nombre de campo*

Nombre del campo de un formulario de texto. El nombre puede hacer referencia a una matriz de campos.

Las opciones son las descritas en la tabla anterior.

**Conceptos relacionados**

- “Componente de formulario” en la página 154
- “Código de datos modificados y propiedad modified” en la página 161
- “Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

**Consulta relacionada**

- “Inicialización de datos” en la página 471
- “Sentencias EGL” en la página 88
- “Propiedades de presentación de campos” en la página 66
- “get next” en la página 597
- “get previous” en la página 602
- “itemsNullable” en la página 395

“Propiedades de elementos SQL” en la página 67

## show

La sentencia **show** presenta un formulario de texto desde un programa principal:

1. Compromete recursos recuperables, cierra archivos y libera bloqueos
2. Opcionalmente, pasa un registro básico para que lo utilice el programa especificado en la cláusula **returning** (si existe) de la sentencia **show**
3. Finaliza el primer programa
4. Presenta el formulario de texto

La sentencia **show** no está disponible en un programa llamada

Si incluye una cláusula **returning** en la sentencia **show**, el entorno de ejecución EGL invoca el programa especificado cuando el usuario pulsa una tecla de evento. Los datos del formulario se asignan al *formulario de entrada* del programa receptor. El registro pasado (no cambiado por la entrada del usuario) se asigna al *registro de entrada* del programa receptor.

Si no incluye una cláusula **returning**, la operación finaliza cuando se presenta el formulario de texto.



### *nombreComponenteFormulario*

Nombre del formulario de texto que es visible para el programa. Para obtener detalles acerca de la visibilidad, consulte el apartado *Referencias a componentes*. Si incluye una cláusula **returning** en la sentencia, el formulario de texto debe ser equivalente al formulario de texto especificado en la propiedad **inputForm** del programa que se invoca.

### **sysVar.transferName**

Variable de sistema que contiene el identificador del programa que debe invocarse. Utilice esta variable para establecer el identificador durante la ejecución.

### *nombreRegistroBásico*

Nombre de un registro de tipo **basicRecord**. El contenido se asigna al *registro de entrada* del programa receptor.

### **Conceptos relacionados**

“Referencias a componentes” en la página 21

### **Consulta relacionada**

“transferName” en la página 939

## transfer

La sentencia EGL **transfer** transfiere el control de un programa principal a otro, finaliza el programa que realiza la transferencia y, opcionalmente, pasa un registro

cuyos datos se aceptan al *registro de entrada* del programa receptor. No puede utilizarse una sentencia **transfer** en un programa llamado.

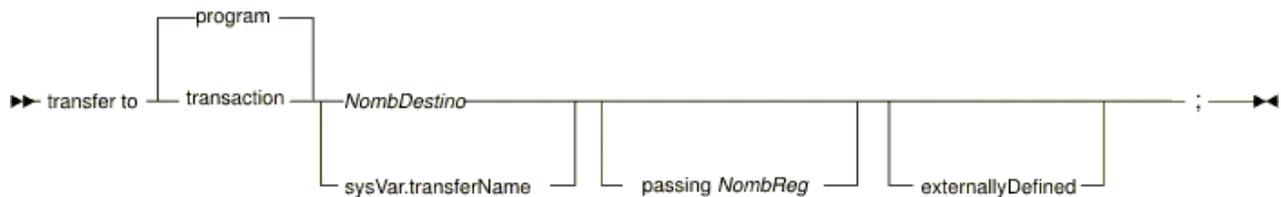
El programa puede transferir el control mediante una sentencia del formato *transferir a transacción* :

- Una transferencia a transacción actúa del siguiente modo:
  - En un programa ejecutado como un programa principal por lotes o de texto Java, el comportamiento depende del valor de la opción **synchOnTrxTransfer** del descriptor de construcción:
    - Si el valor de **synchOnTrxTransfer** es YES, la sentencia transfer compromete los recursos recuperables, cierra archivos, cierra cursores e inicia un programa en la misma unidad de ejecución.
    - Si el valor de **synchOnTrxTransfer** es NO (valor por omisión), la sentencia transfer también inicia un programa en la misma unidad de ejecución, pero no cierra ni compromete recursos, que están disponibles para el programa invocado.
  - En un PageHandler, no puede realizarse una transferencia a una transacción; en lugar de ello, utilice la sentencia **forward**.

El componente de opciones de enlace, elemento **transferLink** no tiene ningún efecto cuando se transfiere el control desde código Java a código Java; de lo contrario, es significativo.

Si transfiere código de control a un código que *no* se ha escrito con EGL o VisualAge Generator, es aconsejable establecer el elemento **transferLink** del componente de opciones de enlace. Establezca la propiedad **linkType** en *externallyDefined*.

Si la ejecución se realiza en modalidad de compatibilidad con VisualAge Generator, puede especificar la opción **externallyDefined** en la sentencia transfer, como ocurre en los programas migrados desde VisualAge Generator; sin embargo, no es aconsejable establecer el valor equivalente en el componente de opciones de enlace. Para obtener detalles acerca de la modalidad de compatibilidad de VisualAge Generator, consulte el apartado *Compatibilidad con VisualAge Generator*.



**program** *nombreDestino* (valor por omisión)

El programa que recibe el control. S

**transaction** *nombreDestino*

La programa que recibe el control, como se ha descrito anteriormente.

**sysVar.transferName**

Función de sistema que contiene un nombre destino que puede establecerse durante la ejecución. Para obtener detalles, consulte la función *sysVar.transferName*.

**passing** *nombreRegistro*

Un registro que se recibe como registro de entrada en el programa destino. El

registro pasado puede ser de cualquier tipo, pero la longitud y los tipos primitivos deben ser compatibles con el registro que recibe los datos. El registro de entrada en el programa destino debe ser de tipo `basicRecord`.

#### Conceptos relacionados

“Compatibilidad con VisualAge Generator” en la página 439

“Creación de alias de nombres” en la página 667

#### Consulta relacionada

“transferName” en la página 939

## try

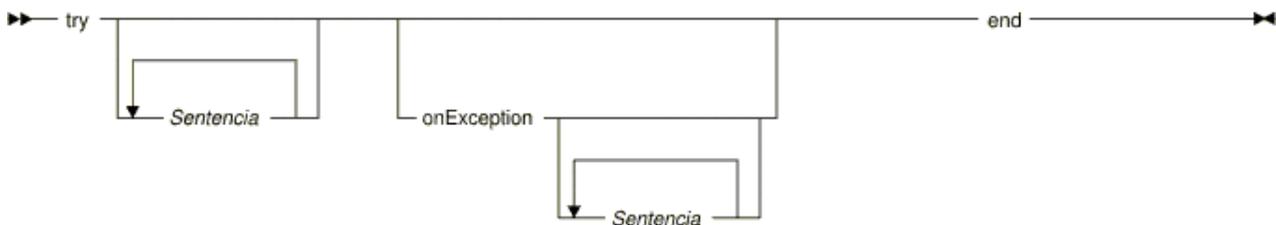
La sentencia **try** de EGL indica que el programa continúa ejecutándose si una sentencia de cualquiera de los siguientes tipos da como resultado un error y se encuentra dentro de la sentencia **try**:

- Una sentencia de entrada/salida (E/S)
- Una invocación de función del sistema
- Una sentencia **call**

Si se produce una excepción, el proceso se reanuda en la primera sentencia del bloque **onException** (si la hay), o en la primera sentencia a continuación del final de la sentencia **try**. No obstante, un error de E/S se maneja solamente si la variable del sistema `VGVar.handleHardIOErrors` está establecida en 1; de lo contrario, el programa visualiza un mensaje (si es posible) y finaliza.

Una sentencia **try** no tiene efecto alguno sobre el comportamiento del entorno de ejecución cuando se produce una excepción en una función o programa invocado desde dentro de la sentencia **try**.

Para conocer otros detalles, consulte *Manejo de excepciones*.



#### *sentencia*

Cualquier sentencia de EGL.

#### **onException**

Un bloque de sentencias que se ejecutan si se produce una condición de excepción.

## while

La palabra clave EGL **while** marca el principio de un conjunto de sentencias que se ejecutan en un bucle. La primera ejecución sólo se produce si una expresión lógica se resuelve en `true` y cada una de las iteraciones subsiguientes depende de la misma prueba. La palabra clave **end** marca el cierre de la sentencia **while**.



*expresión lógica*

Una expresión (una serie de operandos y operadores) que evalúa en true o false

*sentencia*

Una sentencia en el lenguaje EGL

A continuación se ofrece un ejemplo:

```
sum = 0;
i = 1;
while (i < 4)
 sum = inputArray[i] + sum;
 i = i + 1;
end
```

**Tareas relacionadas**

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

**Consulta relacionada**

“Expresiones lógicas” en la página 497

“Sentencias EGL” en la página 88

## Biblioteca (salida generada)

Un componente de biblioteca para salida de Java se genera como una clase Java. El nombre de la clase es el alias de componente (o es el nombre de componente si no se ha especificado alias), pero EGL realiza las sustituciones tal como se describen en *Cómo crear alias de nombres Java*.

**Conceptos relacionados**

“Componente de biblioteca de tipo basicLibrary” en la página 142

“Componente de biblioteca de tipo basicLibrary” en la página 142

“Unidad de ejecución” en la página 742

**Tareas relacionadas**

“Cómo se crean alias de los nombres Java” en la página 669

**Consulta relacionada**

“Componente de biblioteca en formato fuente EGL”

## Componente de biblioteca en formato fuente EGL

Un componente de biblioteca se declara en un archivo EGL, que está descrito en *Formato fuente EGL*.

Este es un ejemplo de un componente de biblioteca:

```
Library CustomerLib3

// Declaraciones Use
```

```

Use StatusLib;

// Declaraciones de datos
exceptionId ExceptionId ;

// Recuperar un cliente para un correo electrónico
// Entrada: cliente, con emailAddress establecido
// Salida: cliente, estado
Function getCustomerByEmail (customer CustomerForEmail, status int)
status = StatusLib.success;
try
get customer ;
onException
exceptionId = "getCustomerByEmail" ;
status = sqlCode ;
end
commit();
end

// Recuperar un cliente para un ID de cliente
// Entrada: cliente, con ID de cliente establecido
// Salida: cliente, estado
Function getCustomerByCustomerId (customer Customer, status int)
status = StatusLib.success;
try
get customer ;
onException
exceptionId = "getCustomerByCustomerId" ;
status = sqlCode ;
end
commit();
end

// Recuperar varios clientes para un correo electrónico
// Entrada: startId
// Salida: customers, status
Function getCustomersByCustomerId
(startId CustomerId, customers Customer[], status int)
status = StatusLib.success;
try
get customers usingKeys startId ;
onException
exceptionId = "getCustomerForEmail" ;
status = sqlCode ;
end
commit();
end

end

```

El diagrama de un componente de biblioteca es el siguiente:



**Library nombreComponenteBiblioteca ... end**

Identifica el componente como un componente de biblioteca y especifica el nombre. Si no establece la propiedad **alias** (como se describe más adelante), el nombre de la biblioteca generada es *libraryPartName*.

Para conocer otras reglas, consulte *Convenios de denominación*.

**type basicLibrary, type nativeLibrary**

Indica el tipo de biblioteca:

- Una biblioteca básica (type basicLibrary) contiene funciones escritas por EGL y valores para tiempo de ejecución en otra lógica de EGL; para conocer más detalles, consulte la sección *Componente de biblioteca de tipo basicLibrary*.
- Una biblioteca nativa (type nativeLibrary) actúa como una interfaz para una DLL externa; para conocer más detalles, consulte la sección *Componente de biblioteca de tipo nativeLibrary*

Por omisión, la biblioteca es de tipo basicLibrary.

*propiedadesBiblioteca*

Las propiedades de biblioteca son las siguientes:

- **alias**
- **allowUnqualifiedItemReferences**

- **callingConvention** (que está disponible solo en bibliotecas de tipo `nativeLibrary`)
- **dllName** (que está disponible solo en bibliotecas de tipo `nativeLibrary`)
- **handleHardIOErrors**
- **includeReferencedFunctions**
- **localSQLScope**
- **messageTablePrefix**
- **throwNrfEofExceptions**

Todas son opcionales:

- **alias** = "*alias*" identifica una serie que se incorpora a los nombres de salida generada. Si no establece la propiedad **alias**, en su lugar se utiliza el nombre del componente programa .
- **allowUnqualifiedItemReferences = no**, **allowUnqualifiedItemReferences = yes** especifica si debe permitirse que el código haga referencia a elementos de estructura, pero excluyendo el nombre del *contenedor*, que es la tabla de datos, registro o formulario que contiene el elemento de estructura. Observe, por ejemplo, el siguiente componente de registro:

```
Record aRecordPart type basicRecord
 10 myItem01 CHAR(5);
 10 myItem02 CHAR(5);
end
```

La siguiente variable está basada en ese componente:

```
myRecord aRecordPart;
```

Si acepta el valor por omisión de **allowUnqualifiedItemReferences** (*no*), debe especificar el nombre de registro al hacer referencia a `myItem01`, como en esta asignación:

```
myValue = myRecord.myItem01;
```

Si establece la propiedad **allowUnqualifiedItemReferences** como *yes*, no obstante, puede evitar especificar el nombre de registro:

```
myValue = myItem01;
```

Se recomienda aceptar el valor por omisión, que ofrece un mejor resultado. Especificando el nombre de contenedor, se reduce la ambigüedad para quien lea su código y para EGL.

EGL utiliza un conjunto de reglas para determinar el área de la memoria a la que hace referencia un nombre de variable o un nombre de elemento. Para conocer detalles, consulte *Referencias a variables y constantes*.

- Tal como se utiliza en una biblioteca de tipo `nativeLibrary`, **callingConvention = I4GL** especifica cómo el tiempo de ejecución de EGL pasa datos entre dos clases de código:

- El código EGL que invoca la función de biblioteca
- La función de la DLL a la que se accede.

El único valor disponible para **callingConvention** es *I4GL*: Para conocer más detalles, consulte la sección *Componente de biblioteca de tipo nativeLibrary*.

- Tal como se utiliza en una biblioteca de tipo `nativeLibrary`, **dllName** especifica el nombre de DLL, que es final; no puede alterarse temporalmente en tiempo de despliegue. Si no especifica un valor para la propiedad de biblioteca **dllName**, debe especificar el nombre de DLL en la propiedad de tiempo de ejecución Java `vgj.defaultI4GLNativeLibrary`.

Para conocer más detalles, consulte la sección *Componente de biblioteca de tipo nativeLibrary*.

- **handleHardIOErrors = yes, handleHardIOErrors = no** establece el valor predeterminado para la variable del sistema **VGVar.handleHardIOErrors**. La variable controla si un programa continúa ejecutándose después de que se haya producido un error grave en una operación de E/S en un bloque try. El valor predeterminado de la propiedad es *yes*, que establece la variable en 1. Para obtener más detalles, consulte *VGVar.handleHardIOErrors* y *Manejo de excepciones*.

- **includeReferencedFunctions = no, includeReferencedFunctions = yes** indica si la biblioteca contiene una copia de cada función que no está ni dentro de la biblioteca ni en una biblioteca a la que accede la biblioteca actual. El valor por omisión es *no*, lo que significa que puede pasar por alto esta propiedad si todas las funciones que deben formar parte de esta biblioteca se encuentran dentro de la misma.

Si la biblioteca utiliza funciones compartidas que no están en ella, la generación sólo es posible si establece la propiedad **includeReferencedFunctions** en *yes*.

- **localSQLScope = yes, localSQLScope = no** indica si los identificadores de los conjuntos de resultados y sentencias preparadas de SQL son locales para el código de biblioteca durante la invocación por un programa o pageHandler, lo que constituye el valor predeterminado. Si acepta el valor *yes*, distintos programas pueden utilizar los mismos identificadores independientemente y el programa o pageHandler que utiliza la biblioteca puede utilizar independientemente los mismos identificadores que se utilizan en la biblioteca.

Si especifica *no*, los identificadores se comparten en toda la unidad de ejecución. Los identificadores creados cuando las sentencias SQL de la biblioteca se invocan están disponibles en otro código que invoca la biblioteca, aunque el otro código puede utilizar **localSQLScope = yes** para bloquear el acceso a esos identificadores. Además, la biblioteca puede hacer referencia a los identificadores creados en el programa invocante o pageHandler, pero solo si las sentencias relacionadas con SQL se hubieran ejecutado ya en el otro código y si el otro código no bloqueara el acceso.

Los efectos de compartir identificadores SQL son los siguientes:

- Puede abrir un conjunto de resultados en un código y obtener filas de ese conjunto en otro
- Puede preparar una sentencia SQL en un código y ejecutar esa sentencia en otra

En cualquier caso, los identificadores disponibles cuando el programa o pageHandler accede a la biblioteca están disponibles cuando el mismo programa o pageHandler accede a la misma o a otra función de la misma biblioteca.

- **msgTablePrefix = "prefijo"** especifica el primero de los cuatro caracteres del nombre de una tabla de datos que se utiliza como tabla de mensajes. (La tabla de mensajes está disponible para los formularios que son la salida de funciones de biblioteca.) Los demás caracteres del nombre corresponden a uno de los códigos de idioma nacional listados en *Componente DataTable en formato fuente EGL*.
- **throwNrfEofExceptions = no, throwNrfEofExceptions = yes** especifica si un error leve provoca el lanzamiento de una excepción. El valor predeterminado es *no*. Para obtener información, consulte la sección *Manejo de excepciones*.

#### *declaraciónUso*

Proporciona fácil acceso a una tabla de datos o biblioteca y es necesario para acceder a formularios de un grupo de formularios. Para conocer detalles, consulte *Declaración Use*.

#### **private**

Indica que la variable, constante o función no está disponible fuera de la biblioteca. Si omite el término **private**, la variable, constante o función estará disponible.

No puede especificar **private** para una función en una biblioteca de tipo `nativeLibrary`.

#### *nombreElementoDatos*

El nombre de un elemento de datos. Para conocer las reglas de denominación, consulte *Convenios de denominación*.

#### *tipoPrimitivo*

El tipo primitivo de un elemento de datos o, en relación con una matriz, el tipo primitivo de un elemento de matriz.

#### *longitud*

La longitud del parámetro o (en relación con una matriz) la longitud de un elemento de matriz. La longitud es un entero que representa el número de caracteres o dígitos en el área de memoria a la que hace referencia *nombreElementoDatos* o (en el caso de una matriz) *nombreMatrizDinámica*.

#### *decimales*

Para un tipo numérico puede especificar *decimales* que es un entero que representa el número de posiciones después de la coma decimal. El número máximo de posiciones decimales es el menor de dos números: 18 o el número de dígitos declarado como *longitud*. La coma decimal no se almacena con los datos.

#### *" máscaraFechaHora"*

Para los tipos `TIMESTAMP` e `INTERVAL` puede especificar *máscaraFechaHora*, que asigna un significado (como por ejemplo "dígito de año") a una posición determinada en el valor de fecha y hora. La máscara no se almacena con los datos.

#### *nombreComponenteElementoDatos*

El nombre de un componente `dataItem` que es visible al programa. Para conocer detalles sobre la visibilidad, consulte *Referencias a componentes*.

El componente actúa como un modelo de formato, tal como se describe en *Typedef*.

#### *nombreRegistro*

El nombre de un registro. Para conocer las reglas de denominación, consulte *Convenios de denominación*.

#### *nombreComponenteRegistro*

El nombre de un componente de registro que es visible al programa. Para conocer detalles sobre la visibilidad, consulte *Referencias a componentes*.

El componente actúa como un modelo de formato, tal como se describe en *Typedef*.

#### *nombreConstante literal*

El nombre y valor de una constante. El valor puede ser una serie entrecomillada o un número. Para conocer las reglas de denominación, consulte *Convenios de denominación*.

#### *propiedadElemento*

Un par de propiedad y valor específico de un elemento, tal como se describe en *Visión general de propiedades y alteraciones temporales de EGL*.

#### *propiedadRegistro*

Un par de propiedad y valor específico de un registro. Para conocer detalles sobre las propiedades disponibles, consulte el tema de referencia para el tipo de registro específico.

Un registro básico no tiene propiedades.

#### *nombreElemento*

El nombre de un elemento de registro cuyas propiedades desea alterar temporalmente. Consulte *Visión general de propiedades y alteraciones temporales de EGL*.

#### *nombreMatriz*

El nombre de una matriz de registros o elementos de datos dinámica o estática. Si utiliza esta opción, los demás símbolos a la derecha (*nombreComponenteElementoDatos*, *tipoPrimitivo* y demás) hacen referencia a cada elemento de la matriz.

#### *tamaño*

El número de elementos de la matriz. Si especifica el número de elementos, la matriz es estática; de lo contrario, la matriz es dinámica.

#### *componenteFunción*

Una función. Ningún parámetro de la función puede ser de un tipo suelto. Para conocer detalles, consulte *Componente de función en formato fuente EGL*.

### **Conceptos relacionados**

“Proyectos, paquetes y archivos EGL” en la página 13

“Componente de biblioteca de tipo basicLibrary” en la página 142

“Componente de biblioteca de tipo basicLibrary” en la página 142

“Visión general de las propiedades de EGL” en la página 64

“Referencias a componentes” en la página 21

“Referencias a variables en EGL” en la página 58

“Typedef” en la página 27

### **Consulta relacionada**

“Componente de registro básico en formato fuente EGL” en la página 379

“Componente DataTable en formato fuente EGL” en la página 473

“Formato fuente EGL” en la página 491

“Manejo de excepciones” en la página 94

“Componente de función en formato fuente EGL” en la página 527

“Componente de registro indexado en formato fuente EGL” en la página 535

“Formulario de entrada” en la página 736

“Registro de entrada” en la página 736

“Valores de error de E/S” en la página 536

“Propiedades de ejecución de Java (detalles)” en la página 540

“Componente de registro MQ en formato fuente EGL” en la página 662

“Tipos primitivos” en la página 34

“Componente de registro relativo en formato fuente EGL” en la página 740

“Componente de registro serie en formato fuente EGL” en la página 743

“Componente de registro SQL en formato fuente EGL” en la página 748

“Declaración use” en la página 954

“handleHardIOErrors” en la página 946

---

## Operador like

En una expresión lógica puede comparar una expresión de texto con otra serie (llamada *criterio like*) por posiciones de caracteres de izquierda a derecha. La utilización de esta característica es parecida a la utilización de la palabra clave SQL **like** en consultas SQL.

A continuación se ofrece un ejemplo:

```
// la variable myVar01 es la expresión de serie
// cuyo contenido se comparará con un criterio like
myVar01 = "abcdef";

// la expresión lógica siguiente evalúa a "true"
if (myVar01 like "a_c%")
;
end
```

El criterio like puede ser un literal o un elemento de tipo CHAR o MBCHAR o un elemento de tipo UNICODE. Puede incluir cualquiera de estos caracteres en el criterio like:

**%** Actúa como un comodín, emparejando cero o más caracteres de la expresión de serie

**\_ (subrayado)**

Actúa como un comodín, emparejando un solo carácter de la expresión de serie

**\** Indica que el carácter siguiente debe compararse con un solo carácter de la expresión de la serie. La barra inclinada invertida (\) se llama *carácter de escape* porque causa un escape del proceso habitual; el carácter de escape no se compara con ningún carácter de la expresión de serie.

El carácter de escape va precede habitualmente un signo de porcentaje (%), un signo de subrayado (\_) u otra barra inclinada invertida.

Cuando utilice la barra inclinada invertida como un carácter de escape (como en el comportamiento por omisión), surge un problema porque EGL utiliza el mismo carácter de escape para permitir la inclusión de unas comillas simples en cualquier expresión de texto. En el contexto de un criterio like debe especificar dos barras inclinadas invertidas porque el texto disponible en tiempo de ejecución es el texto al que le falta la barra inclinada invertida inicial.

Es aconsejable evitar este problema. Especifique otro carácter como el carácter de escape utilizando la cláusula de escape tal como se muestra en un ejemplo posterior. Sin embargo, no puede utilizar unas comillas (") como carácter de escape.

Cualquier otro carácter en *likeCriterion* es un literal comparado con un solo carácter de la *expresión serie*.

El ejemplo siguiente muestra la utilización de una cláusula de escape:

```
// la variable myVar01 es la expresión de serie
// cuyo contenido se comparará con un criterio like
myVar01 = "ab%def";

// la expresión lógica siguiente evalúa a "true"
if (myVar01 like "ab\\%def")
;
end
```

```
// la expresión lógica siguiente evalúa a "true"
if (myVar01 like "ab+%def" escape "+")
;
end
```

#### Consulta relacionada

“Sentencias EGL” en la página 88

“Expresiones lógicas” en la página 497

“Expresiones de texto” en la página 505

---

## Archivo de propiedades de enlace (detalles)

Al generar un programa o envoltura Java llamador, puede especificar que es necesaria información de enlace durante la ejecución. Esa especificación se realiza estableciendo los valores de opción de enlace del programa llamado, del siguiente modo:

- El valor de la propiedad **type** del elemento callLink es remoteCall o.ejbCall; y
- El valor de la propiedad **remoteBind** del elemento callLink es RUNTIME.

Un archivo de propiedades de enlace puede escribirse manualmente, pero EGL un archivo si (además de los valores descritos anteriormente) el usuario genera un programa o envoltura Java con la opción **genProperties** del descriptor de construcción establecida en GLOBAL o PROGRAM.

## Cómo se identifica el archivo de propiedades de enlace durante la ejecución

Si la propiedad **remoteBind** del elemento callLink de un programa llamado se ha establecido en RUNTIME en el componente de opciones de enlace, el archivo de propiedades de enlace se busca durante la ejecución; pero el origen del nombre de archivo es diferente en los programas Java y las envolturas Java:

- Un programa Java comprueba la propiedad de entorno de ejecución Java **cs.linkageOptions.LO**, donde *LO* es el nombre del componente de opciones de enlace utilizado para la generación. Si la propiedad no está presente, el código del entorno de ejecución EGL busca un archivo de propiedades de enlace denominado **LO.properties**. De nuevo, *LO* es el nombre del componente de opciones de enlace utilizado para la generación.

En este caso, si el código del entorno de ejecución EGL busca un archivo de propiedades de enlace pero no puede encontrarlo, se produce un error en la primera sentencia de llamada que requiere la utilización de ese archivo. Para obtener detalles acerca del resultado, consulte el apartado Manejo de excepciones.

- La envoltura Java almacena el nombre del archivo de propiedades de enlace en la variable de objeto programa *callOptions*, que es de tipo CSOCallOptions. El nombre generado del archivo es **LO.properties**, donde *LO* es el nombre del componente de opciones de enlace utilizado para la generación.

En este caso, si la máquina virtual Java busca un archivo de propiedades de enlace pero no puede encontrarlo, el objeto programa lanza una excepción de tipo CSOException.

## Formato del archivo de propiedades de enlace

Cuando se utiliza durante la ejecución, el archivo de propiedades de enlace incluye una serie de entradas destinadas a manejar cada una de las llamadas desde el programa o envoltura Java generados que se están desplegando.

La entrada principal es de tipo `cso.serverLinkage` y puede incluir cualquier par de propiedad y valor que pueda establecerse en un elemento `callLink` del componente de opciones de enlace, con las siguientes excepciones:

- La propiedad **remoteBind** debe ser necesariamente `RUNTIME` y no debe aparecer
- La propiedad **type** no puede ser `localCall`, ya que el enlace de las llamadas locales debe establecerse durante la generación

### Entradas de tipo `cso.serverLinkage`

En el caso más elemental, todas las entradas del archivo de propiedades de enlace son de tipo `cso.serverLinkage`. El formato de la entrada es el siguiente:

```
cso.serverLinkage.nombrePrograma.propiedad=valor
```

*nombrePrograma*

Nombre del programa llamado. Si el programa llamado se genera mediante EGL, el nombre especificado es de un componente de programa.

*propiedad*

Cualquiera de las propiedades adecuadas para un programa Java, excepto las propiedades **remoteBind** y **pgmName**. Para obtener detalles, consulte el apartado correspondiente al *elemento callLink*.

*valor*

Un valor válido para la propiedad especificada.

A continuación figura un ejemplo del programa llamado `XYZ`, donde `xxx` hace referencia a una serie sensible a mayúsculas y minúsculas:

```
cso.serverLinkage.Xyz.type=ejbCall
cso.serverLinkage.Xyz.remoteComType=TCPIP
cso.serverLinkage.Xyz.remotePgmType=EGL
cso.serverLinkage.Xyz.externalName=xxx
cso.serverLinkage.Xyz.package=xxx
cso.serverLinkage.Xyz.conversionTable=xxx
cso.serverLinkage.Xyz.location=xxx
cso.serverLinkage.Xyz.serverID=xxx
cso.serverLinkage.Xyz.parmForm=COMMDATA
cso.serverLinkage.Xyz.providerURL=xxx
cso.serverLinkage.Xyz.luwControl=CLIENT
```

Los valores de literal `TCPIP`, `EGL`, etc. no son sensibles a mayúsculas y minúsculas y son ejemplos de datos válidos.

### Entradas de tipo `cso.application`

Si desea crear una serie de entradas `cso.serverLinkage` que hagan referencia a alguno de varios programas llamados, preceda dichas entradas con una o varias entradas de tipo `cso.application`. En este caso, el objetivo consiste en comparar un solo nombre de aplicación con varios nombres de programa. En las entradas `cso.serverLinkage` subsiguientes, utilizará el nombre de aplicación en lugar de *nombrePrograma*; a continuación, durante la ejecución Java, esas entradas `cso.serverLinkage` manejarán las llamadas a cualquiera de los diversos programas.

El formato de una entrada `cso.application` es el siguiente:

```
cso.application.nombreProgramaLibre.nombreAplicación
```

*nombreProgramaLibre*

Un nombre de programa válido, un asterisco o el principio de un nombre de programa válido seguido de un asterisco. El asterisco es el carácter comodín equivalente a uno o varios caracteres y proporciona una forma de identificar un conjunto de nombres.

Si *nombreProgramaLibre* hace referencia a un programa generado por EGL, cualquier nombre de programa incluido en *nombreProgramaLibre* será el nombre de un componente de programa.

#### *nombreAplicación*

Una serie de caracteres conformes a los convenios de denominación de EGL. El valor de *nombreAplicación* se utiliza en las entradas `cso.serverLinkage` subsiguientes.

El ejemplo siguiente muestra la utilización de un asterisco como carácter comodín. Las entradas `cso.serverLinkage` de este ejemplo manejan las llamadas efectuadas al programa cuyo nombre empiece por *XYZ*:

```
cso.application.Xyz*=myApp
cso.serverLinkage.myApp.type=remoteCall
cso.serverLinkage.myApp.remoteComType=TCPIP
cso.serverLinkage.myApp.remotePgmType=EGL
cso.serverLinkage.myApp.externalName=xxx
cso.serverLinkage.myApp.package=xxx
cso.serverLinkage.myApp.conversionTable=xxx
cso.serverLinkage.myApp.location=xxx
cso.serverLinkage.myApp.serverID=xxx
cso.serverLinkage.myApp.parmForm=COMMDATA
cso.serverLinkage.myApp.luwControl=CLIENT
```

El ejemplo siguiente muestra la utilización de las mismas entradas `cso.serverLinkage` para manejar las llamadas efectuadas a cualquiera de diversos programas, aunque los nombres de dichos programas no empiecen por los mismos caracteres:

```
cso.application.Abc=myApp
cso.application.Def=myApp
cso.application.Xyz=myApp
cso.serverLinkage.myApp.type=remoteCall
cso.serverLinkage.myApp.remoteComType=TCPIP
cso.serverLinkage.myApp.remotePgmType=EGL
cso.serverLinkage.myApp.externalName=xxx
cso.serverLinkage.myApp.package=xxx
cso.serverLinkage.myApp.conversionTable=xxx
cso.serverLinkage.myApp.location=xxx
cso.serverLinkage.myApp.serverID=xxx
cso.serverLinkage.myApp.parmForm=COMMDATA
cso.serverLinkage.myApp.luwControl=CLIENT
```

Si hay varias entradas `cso.application` válidas para un programa, EGL utiliza la primera entrada correcta.

#### **Conceptos relacionados**

“Componente de opciones de enlace” en la página 311

“Archivo de propiedades de enlace” en la página 364

#### **Tareas relacionadas**

“Editar el elemento `callLink` de un componente de opciones de enlace” en la página 314

“Configurar entorno de ejecución J2EE para código generado por EGL” en la página 354

#### **Consulta relacionada**

“Elemento `callLink`” en la página 407

“Manejo de excepciones” en la página 94

“Propiedades de ejecución de Java (detalles)” en la página 540

“Convenios de denominación” en la página 672

---

## Operador matches

En una expresión lógica puede comparar una expresión de serie con otra serie (llamada *criterio match*) por posiciones de caracteres de izquierda a derecha. La utilización de esta característica es parecida a la utilización de *expresiones regulares* en UNIX o Perl.

A continuación se ofrece un ejemplo:

```
// la variable myVar01 es la expresión de serie
// cuyo contenido se comparará con un criterio match
myVar01 = "abcdef";

// la expresión lógica siguiente evalúa a "true"
if (myVar01 matches "a?c*")
;
end
```

El criterio match puede ser un literal o un elemento de tipo CHAR o MBCHAR o un elemento de tipo UNICODE. Puede incluir cualquiera de estos caracteres en el criterio match:

- \* Actúa como un comodín, emparejando cero o más caracteres de la expresión de serie
- ? Actúa como un comodín, emparejando un solo carácter de la expresión de serie
- [ ] Actúa como un delimitador tal que cualquiera de los caracteres entre los dos corchetes es válido como coincidencia para el siguiente carácter de la expresión de serie. El componente siguiente de un criterio match, por ejemplo, indica que a, b o c es válido como coincidencia:

[abc]

- Crea un rango dentro de los corchetes delimitadores tal que cualquier carácter dentro del rango es válido como coincidencia para el carácter siguiente de la expresión de serie. El componente siguiente de un criterio match, por ejemplo, indica que a, b o c es válido como coincidencia:

[a-c]

El guión (-) no tiene un significado especial fuera de los corchetes delimitadores.

- ^ Crea una regla comodín tal que, si el signo de intercalación (^) es el primer carácter dentro de los corchetes delimitadores, cualquier carácter distinto de los caracteres delimitados será una coincidencia válida para el siguiente carácter de la expresión de serie. El componente siguiente de un criterio match, por ejemplo, indica que cualquier carácter distinto de a, b o c es válido como coincidencia:

[^abc]

El signo de intercalación no tiene un significado especial en estos casos:

- Está fuera de los corchetes delimitadores
- Está dentro de los corchetes delimitadores, pero no en la primera posición

- \ Indica que el carácter siguiente debe compararse con un solo carácter de la expresión de la serie. La barra inclinada invertida (\) se llama *carácter de escape* porque causa un escape del proceso habitual; el carácter de escape no se compara con ningún carácter de la expresión de serie.

El carácter de escape precede habitualmente a un carácter que de lo contrario sería relevante en el criterio match; por ejemplo, un asterisco (\*) o un signo de interrogación (?).

Cuando utilice la barra inclinada invertida como un carácter de escape (como en el comportamiento por omisión), surge un problema porque EGL utiliza el mismo carácter de escape para permitir la inclusión de unas comillas simples en cualquier expresión de texto. En el contexto de un criterio match debe especificar dos barras inclinadas invertidas porque el texto disponible en tiempo de ejecución es el texto al que le falta la barra inclinada invertida inicial.

Es aconsejable evitar este problema. Especifique otro carácter como el carácter de escape utilizando la cláusula de escape tal como se muestra en un ejemplo posterior. Sin embargo, no puede utilizar unas comillas (") como carácter de escape.

Cualquier otro carácter en *matchCriterion* es un literal comparado con un solo carácter de la *expresión serie*.

El ejemplo siguiente muestra la utilización de una cláusula de escape:

```
// la variable myVar01 es la expresión de serie
// cuyo contenido se comparará con un criterio match
myVar01 = "ab*def";

// la expresión lógica siguiente evalúa a "true"
if (myVar01 matches "ab*[abcd] [abcde] [^a-e]")
;
end

// la expresión lógica siguiente evalúa a "true"
if (myVar01 matches "ab+*def" escape "+")
;
end
```

#### Consulta relacionada

“Sentencias EGL” en la página 88

“Expresiones lógicas” en la página 497

“Expresiones de texto” en la página 505

---

## Personalización de mensajes para el tiempo de ejecución de Java EGL

Cuando se produce un error en el tiempo de ejecución de Java, por omisión se muestra un mensaje del sistema de EGL pero puede especificar un mensaje personalizado para cada uno de estos mensajes del sistema o para un subconjunto.

Cuando se necesita un mensaje, EGL primero busca en un archivo de propiedades identificado por el usuario en la propiedad `vgj.messages.file` de tiempo de ejecución de Java. El formato del archivo al que se hace referencia es el mismo que el de cualquier archivo de propiedades Java, tal como se describe en el apartado *Archivo de propiedades del programa* y se muestra en el tema actual.

En muchos casos, un mensaje del sistema incluye lugares reservados para las inserciones de mensajes que EGL recupera en tiempo de ejecución. Si el código somete una máscara de fecha no válida para una función del sistema, por ejemplo, el mensaje tiene dos lugares reservados, uno (lugar reservado 0) para la misma máscara de fecha, el otro (lugar reservado 1) para el nombre de la función del sistema. En el formato de archivo de propiedades, la entrada correspondiente al mensaje predeterminado es la siguiente:

VGJ0216E = {0} no es una máscara de fecha válida para {1}.

Puede cambiar el texto del mensaje para que incluya todos o algunos de los lugares reservados en cualquier orden, pero no puede añadir lugares reservados. Los siguientes son ejemplos válidos:

VGJ0216E = La función {1} ha obtenido una máscara de fecha no válida {0}.

VGJ0216E = La función {1} ha obtenido una máscara de fecha no válida.

Se produce un error muy grave si no se puede abrir el archivo identificado en la propiedad `vgj.messages.file`.

Para conocer los detalles de los números de los mensajes y de su significado, consulte el apartado *Códigos de error de tiempo de ejecución de Java EGL*.

Encontrará más detalles en la documentación del lenguaje Java:

- Para conocer más detalles acerca de cómo se procesan los mensajes y de qué contenido es válido, consulte la documentación para la clase Java `java.text.MessageFormat`.
- Para conocer más detalles acerca de cómo manejar los caracteres que no pueden representarse directamente en la codificación de caracteres ISO 8859-1 (que se utiliza siempre en los archivos de propiedades), consulte la documentación para la clase Java `java.util.Properties`.

#### Conceptos relacionados

“Archivo de propiedades del programa” en la página 350

#### Consulta relacionada

“Código de error de ejecución de Java EGL” en la página 959

“Propiedades de ejecución de Java (detalles)” en la página 540

---

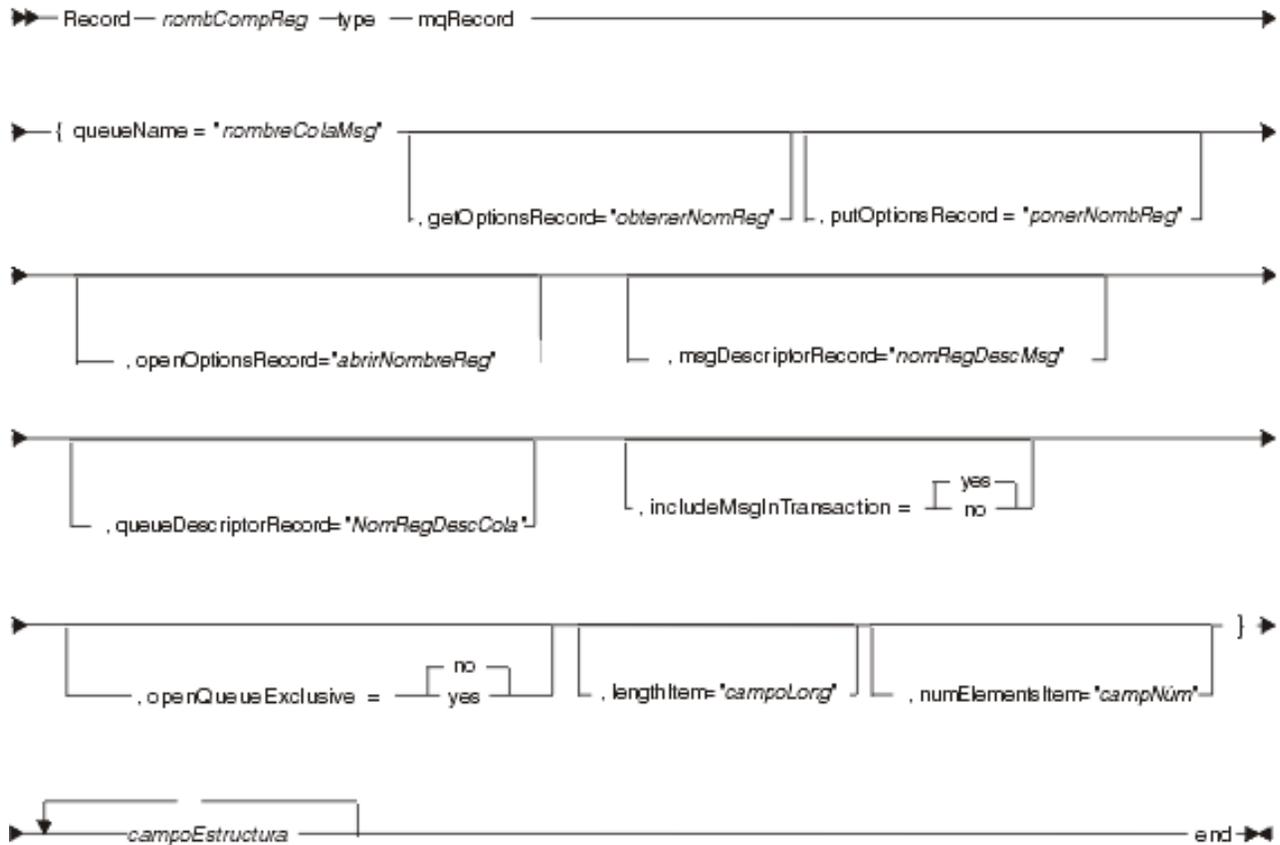
## Componente de registro MQ en formato fuente EGL

Puede declarar componentes de registro MQ en un archivo fuente de EGL. Para obtener una visión general de ese archivo, consulte el apartado *Formato fuente EGL*. Para obtener una visión general de la forma en que EGL interactúa con MQSeries, consulte el apartado *Soporte de MQSeries*.

A continuación se ofrece un ejemplo de componente de registro MQ:

```
Record myMQRecordPart type mqRecord
{
 queueName = "myQueue"
}
10 myField01 CHAR(2);
10 myField02 CHAR(78);
end
```

El diagrama de sintaxis de un componente de registro MQ es el siguiente:



**Record nombreComponenteRegistro mqRecord**

Identifica el componente como de tipo mqRecord y especifica el nombre. Para conocer las normas, consulte el apartado *Convenios de denominación*.

**queueName = "nombreColaMsg"**

El nombre de la cola de mensajes, que es el nombre de la cola lógica y, generalmente, no el nombre de la cola física. Para obtener detalles acerca del formato de la entrada, consulte el apartado *Propiedades de registros MQ*.

**getOptionsRecord = "obtenerNombreRegistro"**

Identifica una variable de programa (un registro básico) utilizada como registro de opciones get. Para obtener detalles, consulte el apartado *Registros de opciones para registros MQ*. Esta propiedad era anteriormente la propiedad **getOptions**.

**putOptionsRecord = "ponerNombreRegistro"**

Identifica una variable de programa (un registro básico) utilizada como registro de opciones put. Para obtener detalles, consulte el apartado *Registros de opciones para registros MQ*. Esta propiedad era anteriormente la propiedad **putOptions**.

**openOptionsRecord = "abrirNombreRegistro"**

Identifica una variable de programa (un registro básico) utilizada como registro de opciones open. Para obtener detalles, consulte el apartado *Registros de opciones para registros MQ*. Esta propiedad era anteriormente la propiedad **openOptions**.

**msgDescriptorRecord = "nombreRegDescMsg"**

Identifica una variable de programa (un registro básico) utilizada como descriptor de mensajes. Para obtener detalles, consulte el apartado *Registros de opciones para registros MQ*. Esta propiedad era anteriormente la propiedad **msgDescriptor**.

**queueDescriptorRecord** = "*nombreRegDescColas*"

Identifica una variable de programa (un registro básico) utilizada como descriptor de colas. Para obtener detalles, consulte el apartado *Registros de opciones para registros MQ*. Esta propiedad era anteriormente la propiedad **queueDescriptor**.

**includeMsgInTransaction** = **yes**, **incluirMsgEnTransacción** = **no**

Si esta propiedad se establece en *yes* (valor por omisión), cada uno de los mensajes específicos de registro se incorpora en una transacción, y el código puede comprometer o retrotraer dicha transacción. Para obtener detalles acerca de las implicaciones de su elección, consulte el apartado *Soporte de MQSeries*.

**openQueueExclusive** = **no**, **openQueueExclusive** = **yes**

Si esta propiedad se establece en *yes*, el código tiene la capacidad exclusiva de leer la cola de mensajes; de lo contrario, otros programas podrán leer la cola. El valor por omisión es *no*. Esta propiedad es equivalente a la opción de MQSeries MQOO\_INPUT\_EXCLUSIVE.

**lengthItem** = "*campoLongitud*"

El campo de longitud, tal como se describe en *Propiedades de registro MQ*.

**numElementsItem** = "*campoNúmElementos*"

El campo de número de elementos, tal como se describe en *Propiedades de registro MQ*.

*campoEstructura*

Un campo de estructura, como se describe en la sección *Elemento de estructura en formato fuente EGL*.

### **Conceptos relacionados**

"Proyectos, paquetes y archivos EGL" en la página 13

"Referencias a componentes" en la página 21

"Soporte de MQSeries" en la página 265

"Componentes" en la página 17

"Componentes de registro" en la página 132

"Referencias a variables en EGL" en la página 58

"Typedef" en la página 27

### **Tareas relacionadas**

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

### **Consulta relacionada**

"Matrices" en la página 74

"Componente DataItem en formato fuente EGL" en la página 472

"Formato fuente EGL" en la página 491

"Componente de función en formato fuente EGL" en la página 527

"Componente de registro indexado en formato fuente EGL" en la página 535

"Propiedades de registros MQ" en la página 665

"Convenios de denominación" en la página 672

"Registros de opciones para registros MQ" en la página 665

"Tipos primitivos" en la página 34

"Componente de programa en formato fuente EGL" en la página 728

"Componente de registro relativo en formato fuente EGL" en la página 740

"Componente de registro serie en formato fuente EGL" en la página 743

"Componente de registro SQL en formato fuente EGL" en la página 748

"Elemento de estructura en el formato fuente de EGL" en la página 752

---

## Propiedades de registros MQ

Esta página describe las siguientes propiedades de registros MQ:

- Nombre de cola
- Incluir mensaje en transacción
- Abrir cola de entrada para uso exclusivo

Para obtener detalles acerca de otras propiedades, consulte estas páginas:

- Registros de opciones para registros MQ
- Propiedades que dan soporte a registros de longitud variable

### Nombre de cola

El *nombre de cola* es obligatorio y hace referencia al nombre de cola lógico, que no puede tener más de 8 caracteres. Para obtener detalles acerca del significado de la entrada, consulte el apartado *Palabras clave EGL relacionadas con MQSeries*.

### Incluir mensaje en transacción

Si se establece, *Incluir mensaje en transacción* intercala cada uno de los mensajes específicos de registro en una transacción, y el código puede comprometer o retrotraer dicha transacción.

Para obtener detalles acerca de las implicaciones de su elección, consulte el apartado *Soporte de MQSeries*.

### Abrir cola de entrada para uso exclusivo

Si establece *Abrir cola de entrada para uso exclusivo*, el código tiene la capacidad exclusiva de leer la cola de mensajes; de lo contrario, otros programas podrán leer la cola. Esta propiedad es equivalente a la opción de MQSeries `MQOO_INPUT_EXCLUSIVE`.

#### Conceptos relacionados

“Palabras clave EGL relacionadas con MQSeries” en la página 268

“Soporte de MQSeries” en la página 265

“Tipos de registros y propiedades” en la página 135

#### Consulta relacionada

“Registros de opciones para registros MQ”

“Propiedades que dan soporte a registros de longitud variable” en la página 737

## Registros de opciones para registros MQ

Cada registro MQ está asociado con cinco *registros de opciones*, que EGL utiliza como argumentos en las llamadas ocultas a MQSeries:

- Registro de opciones get (MQGMO)
- Registro de opciones put (MQPMO)
- Registro de opciones open (MQOO: un registro con un elemento de estructura)
- Registro descriptor de mensajes (MQMD)
- Registro descriptor de cola (MQOD)

Al especificar un registro de opciones como propiedad de un registro MQ, está haciendo referencia a una variable que utiliza un componente de registro de almacenamiento de trabajo (como MQOD) como definición de tipo (typeDef). El componente reside en un archivo EGL que se suministra con el producto, como se

describe en el apartado *Soporte de MQSeries*. En lugar de utilizar el componente de registro tal cual, puede copiarlo en su propio archivo EGL y personalizarlo.

Si no indica que se está utilizando un componente de opciones determinado, EGL construye un registro por omisión y le asigna valores, como se describe en las siguientes secciones. Los registros de opciones por omisión no están disponibles, sin embargo, si accede a MQSeries sin utilizar registros MQ.

### Registro de opciones get

Puede crear un registro de opciones get basado en MQGMO (Opciones de mensaje Get) de MQSeries, que es un argumento de las llamadas MQGET de MQSeries. Si no declara un registro de opciones get, EGL crea automáticamente un valor por omisión denominado MQGMO y el programa generado hace lo siguiente:

- Inicializa el registro de opciones get con los valores indicados al principio de *Inicialización de datos*
- Establece OPTIONS en MQGMO\_SYNCPOINT o MQGMO\_NO\_SYNCPOINT, dependiendo de si ha establecido la propiedad de registro MQ *Incluir mensaje en transacción*

### Registro de opciones put

Puede crear un registro de opciones put basado en MQPMO (Opciones de mensaje Put) de MQSeries, que es un argumento de las llamadas MQPUT de MQSeries. Si no declara un registro de opciones put, EGL crea automáticamente un valor por omisión denominado MQPMO y el programa generado hace lo siguiente:

- Inicializa el registro de opciones put con los valores indicados al principio de *Inicialización de datos*
- Establece OPTIONS en MQPMO\_SYNCPOINT o MQPMO\_NO\_SYNCPOINT, dependiendo de si ha establecido la propiedad de registro MQ *Incluir mensaje en transacción*

### Registro de opciones open

El contenido del registro de opciones open determina el valor del parámetro Opciones utilizado en las llamadas al mandato MQSeries MQOPEN o MQCLOSE. El componente de registro de opciones open (MQOO) está disponible, pero si no declara un registro basado en ese componente, EGL crea automáticamente uno por omisión denominado MQOO del siguiente modo:

- En un MQOPEN que se invoca debido a una sentencia EGL add, el programa generado establece MQOO.OPTIONS en este valor:  
MQOO\_OUTPUT + MQOO\_FAIL\_IF QUIESCING
- En un MQOPEN que se invoca debido a una sentencia EGL scan, el programa generado establece MQOO.OPTIONS en el siguiente valor cuando la opción de propiedad de registro de cola de mensajes *Abrir cola de entrada para uso exclusivo* está en vigor:  
MQOO\_INPUT\_EXCLUSIVE + MQOO\_FAIL\_IF QUIESCING
- En un MQOPEN que se invoca debido a una sentencia EGL scan, el programa generado establece MQOO.OPTIONS en el siguiente valor cuando la opción de propiedad de registro de cola de mensajes *Abrir cola de entrada para uso exclusivo* no está en vigor:  
MQOO\_INPUT\_SHARED + MQOO\_FAIL\_IF QUIESCING
- En un MQCLOSE que se invoca debido a una sentencia EGL close, el programa generado establece MQOO.OPTIONS en este valor:  
MQCO\_NONE

## Registro descriptor de mensajes

Puede crear un registro descriptor de mensajes basado en MQMD (Descriptor de mensajes) de MQSeries, que es un parámetro de las llamadas MQGET y MQPUT. Si no declara un registro descriptor de mensajes, EGL crea automáticamente un valor por omisión denominado MQMD y lo inicializa con los valores indicados en *Inicialización de datos*.

## Registro descriptor de cola

Puede crear un registro de descriptor de colas basado en MQOD (Descriptor de objeto MQSeries) que es un argumento de las llamadas MQOPEN y MQCLOSE de MQSeries. Si no declara un registro descriptor de cola, EGL crea automáticamente un valor por omisión denominado MQOD y el programa generado hace lo siguiente:

- Inicializa el registro descriptor de cola con los valores indicados al principio de *Inicialización de datos*
- Establece el valor OBJECTTYPE de ese registro en MQOT\_Q
- Establece el valor OBJECTMGRNAME en el nombre de gestor de colas especificado en la palabra de sistema **record.resourceAssociation**; pero si **record.resourceAssociation** no hace referencia al nombre del gestor de colas, OBJECTQMGRNAME no tiene ningún valor
- Establece el valor OBJECTNAME en el nombre de cola de **record.resourceAssociation**

### Conceptos relacionados

“Llamadas directas a MQSeries” en la página 270

“Palabras clave EGL relacionadas con MQSeries” en la página 268

“Soporte de MQSeries” en la página 265

### Consulta relacionada

“Inicialización de datos” en la página 471

“recordName.resourceAssociation” en la página 859

“Propiedades de registros MQ” en la página 665

---

## Creación de alias de nombres

Si utiliza un nombre que no es válido en la salida Java, el generador crea y utiliza un alias para el nombre en el código generado, por cualquiera de las siguientes razones:

- Diferencias en los caracteres permitidos del identificador
- Diferencias en las limitaciones de longitud
- Diferencias en el soporte para caracteres en mayúsculas y minúsculas
- Utilización de una palabra que es una palabra reservada en el lenguaje generado
- Utilización de una palabra que entra en conflicto con la sintaxis del alias del nombre (por ejemplo, se crea un alias de **class\$** porque **class\$** es el alias de **class** en la generación Java)

Se puede generar un alias sustituyendo un conjunto válido de caracteres por un carácter no válido, truncando nombres que son demasiado largos, añadiendo un prefijo o sufijo a un nombre o generando un nombre completamente distinto, como por ejemplo **EZE00123**.

### Conceptos relacionados

### Tareas relacionadas

“Crear un componente de programa de EGL” en la página 138

### Consulta relacionada

“Cómo se crean alias de los nombres Java” en la página 669

“Cómo se crean alias de los nombres de envoltura Java” en la página 670

“Convenios de denominación” en la página 672

## Cambios en identificadores EGL de archivos JSP y beans Java generados

Puede asignar nombres a funciones de PageHandler, registros y elementos de acuerdo con las reglas detalladas en los *Convenios de denominación*. Sin embargo, EGL utiliza una variación de esos nombres al crear identificadores Java en archivos JSP y en el bean Java derivado de un PageHandler. Debe estar al tanto de esas variaciones si utiliza la pestaña de código fuente para editar un archivo JSP, si utiliza la vista Propiedades o si trabaja fuera de las herramientas habilitadas para EGL.

Las variaciones son las siguientes:

- Las letras *EGL* preceden a los nombres de los registros del PageHandler, de los elementos y de las funciones. El objetivo de esta variación consiste en protegerle de los errores que pueden producirse en el entorno de tiempo de ejecución Java como resultado de las diferencias entre la especificación de bean Java bean y los convenios de denominación en EGL.
- En varias situaciones, se añade un sufijo al nombre de una variable enlazado a una especie determinada de control de salida:
  - Si enlaza un elemento a un recuadro de selección booleano, el identificador Java incluye el sufijo *AsBoolean*
  - Si enlaza un elemento a un control de selección (un recuadro de lista, un recuadro combinado, un grupo de botones de selección, o un grupo de recuadros de selección) y hace referencia al elemento en el código `selectItems` de JavaServer Faces, el identificador Java incluye el sufijo *AsSelectItemsList*
  - Si enlaza un elemento a un recuadro de selección en una tabla de datos JavaServer Faces (específicamente, si se hace referencia al elemento en un código `inputRowSelect`), el identificador Java incluye el sufijo *AsIntegerArray*

Aparte de las variaciones indicadas anteriormente, EGL intenta crear un identificador que coincida exactamente con el nombre del PageHandler.

Fíjese en el PageHandler *myJSP*, que incluye la variable *myItem*. Si enlaza esa variable a un recuadro de selección Booleano, el archivo JSP hace referencia a la propiedad del bean Java *myJSP.EGLmyItemAsBoolean* y las funciones de obtención y establecimiento del bean Java se denominan de la forma siguientes:

- *getEGLmyItemAsBoolean*
- *setEGLmyItemAsBoolean*

El código fuente del código del recuadro de selección booleano del archivo JSP es el siguiente:

```
<h:selectBooleanCheckbox styleClass="selectBooleanCheckbox"
 id="checkbox1" value="#{myJSP.EGLmyItemAsBoolean}">
</h:selectBooleanCheckbox>
```

EGL evita generar un nombre que no sería válido en Java; para conocer más detalles, consulte la sección *Establecimiento de alias de nombres Java*.

#### Conceptos relacionados

“PageHandler” en la página 194

#### Tareas relacionadas

“Crear un campo de EGL y asociarlo con un JSP Faces” en la página 199

“Asociar un registro EGL con un JSP Faces” en la página 200

“Utilizar la vista Edición rápida para el código de PageHandler” en la página 202

#### Consulta relacionada

“Cómo se crean alias de los nombres Java”

“Convenios de denominación” en la página 672

“Soporte de Page Designer para EGL” en la página 192

## Cómo se crean alias de los nombres Java

Al dar un nombre a un componente, dicho nombre debe ser un identificador Java válido, excepto que puede utilizar un guión o un signo menos (-) en un nombre de componente. Sin embargo, un guión no puede ser el primer carácter de un nombre de componente.

Si elige un nombre que es una palabra clave Java o un nombre que contiene un signo de dólar (\$), un guión o un signo menos, el nombre de componente no coincidirá con el nombre que figura en la salida generada. Un mecanismo de creación de alias añade automáticamente un signo de dólar al final de cada nombre de componente que es una palabra clave Java. Si especifica un nombre que contiene uno o varios signos de dólar o guiones, el mecanismo de alias sustituirá cada símbolo por un valor Unicode como se indica a continuación:

\$ \$0024

- \$002d

Por ejemplo, un elemento denominado **class** toma el alias de **class\$** y un elemento denominado **class\$** toma el alias de **class\$0024**.

Se conservan las mayúsculas o minúsculas que utilice para declarar un nombre de componente. Los programas XYZ y xyz se generan en XYZ.java y xyz.java respectivamente. En Windows 2000/NT/XP, si genera en el mismo directorio componentes cuyo nombre sólo difiere en las mayúsculas y minúsculas, los archivos antiguos se sobrescribirán.

Los nombres de paquete EGL se convierten siempre nombres de paquete Java en minúsculas.

Finalmente, si el nombre de un programa, PageHandler o biblioteca coincide con el nombre de una clase del paquete de sistema Java java.lang, se añade un signo de dólar al final del nombre de clase: Object se convierte en Object\$, Error se convierte en Error\$, etc.

Para obtener detalles sobre cómo EGL crea identificadores Java en campos JSP y en el bean Java derivado de un PageHandler, consulte la sección *Cambios en identificadores EGL en archivos JSP y beansJava generados*.

### Conceptos relacionados

“Creación de alias de nombres” en la página 667

### Consulta relacionada

“Cambios en identificadores EGL de archivos JSP y beans Java generados” en la página 668

## Cómo se crean alias de los nombres de envoltura Java

El generador de EGL aplica las siguientes reglas a los alias de nombres de envoltura de Java:

1. Si el nombre de EGL está en mayúsculas, conviértalo a minúsculas
2. Si el nombre es un nombre de clase o un nombre de método, ponga el primer carácter en mayúsculas. (Por ejemplo, el método de obtención para x es `getX()` no `getx()`.)
3. Suprima todos los subrayados (`_`) y guiones (`-`). (Los guiones son válidos en nombres EGL si utiliza la modalidad de compatibilidad de VisualAge Generator.) Si a continuación del subrayado o guión hay una letra, cambie ese carácter a mayúsculas.
4. Si el nombre es un nombre calificado que utiliza un punto (`.`) como separador, sustituya cada punto por un subrayado y añada un subrayado al principio del nombre.
5. Si el nombre contiene un signo de dólar (`$`), sustitúyalo por dos subrayados y añada un subrayado al principio del nombre.
6. Si un nombre es una palabra clave Java, añada un subrayado al principio del nombre.
7. Si el nombre es `*` (un asterisco, que representa un elemento de relleno), redenomine el primer asterisco como **Filler1**, el segundo asterisco como **Filler2** y así sucesivamente.

Además, se aplican normas especiales a los nombres de clase de envoltura Java para envolturas de programa, envolturas de registro y elementos de matriz subestructurada. Las secciones restantes tratan estas normas y ofrecen un ejemplo. En general, si existen conflictos de denominación entre campos dentro de una clase de envoltura generada, se utiliza el nombre calificado para determinar los nombres de clase y variable. Si el conflicto sigue sin resolverse, se emite una excepción durante la generación.

### Clase de envoltura de programa

Las envolturas de parámetros de registro se denominan utilizando las normas anteriores aplicadas al nombre de definición de tipo. Si el nombre de clase de envoltura de registro está en conflicto con el nombre de clase de programa o el nombre de clase de envoltura de programa, se añade **Record** al final del nombre de clase de envoltura de registro.

Las normas para los nombres de variable son las siguientes:

1. La variable de parámetro de registro se denomina utilizando las normas anteriores aplicadas al nombre de parámetro. Por consiguiente, los métodos `get()` y `set()` contienen estos nombres en lugar del nombre de clase.
2. Los métodos `get` y `set` se denominan `get` o `set` seguidos del nombre de parámetro aplicando las normas anteriores.

### Clase de envoltura de registro

Las normas para los nombres de clase de elementos de matriz subestructurada son las siguientes:

1. El elemento de matriz subestructurada se convierte en una clase interna de la clase de envoltura de registro, y el nombre de clase se deriva aplicando las normas anteriores al nombre de elemento. Si el nombre de clase está en conflicto con el nombre de clase de elemento de matriz subestructurada que la contiene, se añade **Structure** al nombre de clase de elemento.
2. Si hay nombres de clase de elemento en conflicto, se utilizarán los nombres de elemento calificados.

Las normas para los nombres de método **get** y **set** son las siguientes:

1. Los métodos se denominan **get** o **set** seguidos del nombre de elemento aplicando las normas anteriores.
2. Si hay nombres de elemento en conflicto, se utilizarán los nombres de elemento calificados.

### Clase de elementos de matriz subestructurada

Las normas para los nombres de clase de elementos de matriz subestructurada son las siguientes:

1. El elemento de matriz subestructurada se convierte en una clase interna de la clase de envoltura generada para el elemento de matriz subestructurada continente, y el nombre de clase se deriva aplicando las normas anteriores al nombre de elemento.
2. Si el nombre de clase está en conflicto con el nombre de clase de elemento de matriz subestructurada continente, se añade **Structure** al nombre de clase de elemento.

Las normas para los nombres de método **get** y **set** son las siguientes:

1. Los métodos se denominan **get** o **set** seguidos del nombre de elemento aplicando las normas anteriores.
2. Si hay nombres de elemento en conflicto, se utilizarán los nombres de elemento calificados.

### Ejemplo

El siguiente programa y salida generada de ejemplo muestran qué debe esperarse durante la generación de envoltura:

#### Programa de ejemplo:

```
Program WrapperAlias(param1 RecordA)
```

```
end
```

```
Record RecordA type basicRecord
```

```
 10 itemA CHAR(10)[1];
```

```
 10 item_b CHAR(10)[1];
```

```
 10 item$C CHAR(10)[1];
```

```
 10 static CHAR(10)[1];
```

```
 10 itemC CHAR(20)[1];
```

```
 15 item CHAR(10)[1];
```

```
 15 itemD CHAR(10)[1];
```

```
 10 arrayItem CHAR(20)[5];
```

```
 15 innerItem1 CHAR(10)[1];
```

```
 15 innerItem2 CHAR(10)[1];
```

```
end
```

## Salida generada:

Nombres de la salida generada

Salida	Nombre
Clase de envoltura de programa	<b>WrapperAliasWrapper</b> , que contiene un campo <b>param1</b> , que es una instancia de la clase de envoltura de registro <b>RecordA</b>
Clases de envoltura de parámetro	<b>RecordA</b> , accesible mediante los siguientes métodos: <ul style="list-style-type: none"><li>• <b>getItemA</b> (desde itemA)</li><li>• <b>getItemB</b> (desde el primer item-b)</li><li>• <b>get_Item_C</b> (desde item\$C)</li><li>• <b>get_Static</b> (desde static)</li><li>• <b>get_ItemC_itemB</b> (desde itemB en itemC)</li><li>• <b>getItemD</b> (desde itemD)</li><li>• <b>getItemArray</b> (desde arrayItem)</li></ul> <b>ArrayItem</b> es una clase interna de <b>RecordA</b> que contiene campos a los que se puede acceder mediante <b>getInnerItem1</b> y <b>getInnerItem2</b> .

### Conceptos relacionados

“Compatibilidad con VisualAge Generator” en la página 439

“Envoltura Java” en la página 301

“Creación de alias de nombres” en la página 667

### Tareas relacionadas

“Generar envolturas Java” en la página 300

### Consulta relacionada

“Clases de envoltura Java” en la página 551

“Convenios de denominación”

“Salida de la generación de envoltura Java” en la página 676

---

## Convenios de denominación

Esta página describe las normas de denominación de componentes y variables y de asignación de propiedades tales como **file name**. Para obtener detalles acerca de cómo los componentes lógicos pueden hacer referencia a áreas de memoria, consulte los temas *Referencias a variables y constantes* y *Matrices*.

En EGL hay tres categorías de identificador:

- Nombres de componentes y variables de EGL, como se describe más adelante.
- Nombres de recursos externos que se especifican como valores de propiedad en declaraciones de componentes o variables. Estos nombres representan casos especiales y los convenios de denominación dependen de los convenios del sistema de ejecución.
- Nombres de paquete de EGL, tales como `com.mycom.mypack`. En este caso, cada secuencia de caracteres está separada de la siguiente por un punto y cada secuencia sigue el convenio de denominación para un nombre de componente EGL. Encontrará los detalles sobre la relación entre nombres de paquete y estructura de archivos en *Proyectos, paquetes y archivos EGL*.

Un nombre de componente o variable de EGL es una serie de 1 a 128 caracteres. Excepto en los casos indicados, un nombre debe empezar por una letra o

subrayado Unicode y puede incluir letras Unicode adicionales, así como dígitos y símbolos de moneda. Se aplican otras restricciones:

- Los primeros caracteres no pueden ser EZE en ninguna combinación de mayúsculas y minúsculas
- Un nombre no puede contener blancos intercalados ni ser una palabra reservada EGL

Se aplican consideraciones especiales a los componentes:

- En un componente de registro, el nombre de un archivo lógico o cola no puede tener más de 8 caracteres.
- En varios componentes, el *alias* se incorpora a los nombres de archivos de salida generados y clases Java. Si no se especifica el nombre externo, se utiliza el nombre del componente de programa, pero se trunca (si es necesario) al llegar al número máximo de caracteres permitidos en el entorno de ejecución.

Si su código es compatible con VisualAge Generator, las normas siguientes también son aplicables a los nombres de componentes y variables pero no tienen ningún efecto sobre los nombres de paquete:

- El carácter inicial de un nombre puede ser un signo @
- Las caracteres siguientes pueden incluir signos @, guiones (-) y almohadillas (#)

#### Conceptos relacionados

“Compatibilidad con VisualAge Generator” en la página 439

“Proyectos, paquetes y archivos EGL” en la página 13

“Creación de alias de nombres” en la página 667

“Referencias a variables en EGL” en la página 58

#### Consulta relacionada

“Matrices” en la página 74

“Cambios en identificadores EGL de archivos JSP y beans Java generados” en la página 668

“Palabras reservadas EGL” en la página 486

“Límites de sistema EGL” en la página 494

---

## Operadores y precedencia

La tabla siguiente indica los operadores EGL por orden de precedencia descendente. Excepto para el signo único más (+), menos (-) y not (!), cada operador funciona con dos operandos.

Operadores (separados por comas)	Tipo de operador	Significado
+, -	Numérico, único	Más (+) o menos (-) único es un signo que se coloca antes de un operando o expresión entre paréntesis, no un operador entre dos expresiones.
**	Numérico	** es el operador <i>toThePowerOfInteger</i> , que acepta un número para la potencia especificada. Por ejemplo $c = a^{**}b$ resultará en la asignación a c del valor de $(a^b)$ . El primer operando (a en el ejemplo anterior) no puede tener un valor negativo. El segundo operando (b en el ejemplo anterior) debe ser un entero o un campo numérico con precisión 0. El segundo operando puede ser positivo, negativo o 0.

Operadores (separados por comas)	Tipo de operador	Significado
*, /, %	Númérico	La multiplicación (*) y la división de enteros (/) tienen la misma precedencia. La división de enteros conserva un valor fraccionario, si existe: por ejemplo, 7/5 da como resultado 1.4.  % es el operador <i>remainder</i> , que se resuelve en el módulo cuando el primero de dos operandos o expresiones numéricas se divide por el segundo; por ejemplo, 7%5 da como resultado 2.
+, -	Númérico	La adición (+) y la sustracción (-) son de igual precedencia.
=	Númérico o serie	= es el operador <i>assignment</i> , que copia un valor numérico o de carácter de una expresión u operando en un operando.
!	Lógico, único	! es el operador <i>not</i> , que se resuelve en un valor booleano (true o false) opuesto al valor de una expresión lógica que sigue inmediatamente. Dicha expresión subsiguiente debe estar entre paréntesis.
==, !=, <, >, <=, >=, in, is, not	Lógico para comparación	Los operadores lógicos utilizados para comparación tienen la misma precedencia y se describen en la página relativa a las expresiones lógicas. Cada operador se resuelve en true o false.
&&	Lógico	&& es el operador <i>and</i> , que significa "ambas deben ser true." El operador se resuelve en true si la expresión lógica que precede al operador es true y si la expresión lógica que sigue al operador es true; de lo contrario, && se resuelve en false.
	Lógico	es el operador <i>or</i> , que significa "una, la otra o ambas." El operador se resuelve en true si la expresión lógica que precede al operador es true, si la expresión lógica que sigue al operador es true o si ambas son true; de lo contrario,    se resuelve en false.

Puede alterar temporalmente la precedencia habitual (también llamada *orden de operaciones*) utilizando paréntesis para separar una expresión de otra. Las operaciones que tienen la misma precedencia en una expresión se evalúan de izquierda a derecha.

#### Consulta relacionada

- "operador in" en la página 532
- "Expresiones lógicas" en la página 497
- "Expresiones numéricas" en la página 504
- "Tipos primitivos" en la página 34
- "Expresiones de texto" en la página 505

## Salida de la generación de programa Java

La salida de la generación de un programa de servidor Java es la siguiente:

- Un plan de construcción, si se ha omitido la opción del descriptor de construcción **genProject**
- Código fuente Java (consulte el apartado *Programa, PageHandler y biblioteca Java*)
- Objetos relacionados necesarios para preparar y ejecutar el programa (consulte el apartado *Programa, PageHandler y biblioteca Java*)
- Archivo de entorno J2EE
- Archivo de propiedades del programa
- Un archivo de resultados, si se omite **genProject**

Puede utilizar el generador EGL para generar programas Java enteros. Los programas y los registros se generan como clases Java independientes. Las funciones se generan como métodos en el programa. Los elementos de datos y los elementos de estructura se generan como campos del registro o clase de programa a la que pertenecen.

La tabla siguiente muestra los nombres de los diversos tipos de componentes Java generados:

Nombres de los componentes Java generados

Tipo y nombre de componente	Qué se genera
Programa denominado P	Una clase denominada P en P.java
Función denominada F en el programa P	Un método de la clase P denominado \$funcF en P.java
Un registro denominado R	Una clase denominada EzeR en EzeR.java
Un registro básico denominado R, parámetro de Función F	Una clase denominada Eze\$paramR en Eze\$paramR.java
Componente de opciones de enlace denominado L	Archivo de propiedades de enlace denominado L.properties
Un biblioteca denominada Lib	Una clase denominada Lib en Lib.java
DataTable denominada DT	Una clase denominada EzeDT en EzeDT.java
Un formulario denominado F	Una clase denominada EzeF en EzeF.java
FormGroup denominado FG	Una clase denominada FG en FG.java

1. Para los tipos de componente indicados, es posible que existan dos o más componentes con el mismo nombre. En ese caso, el nombre del segundo tendrá un sufijo adicional, \$v2. El nombre del tercero tendrá un sufijo \$v3, el cuarto tendrá \$v4, etc.

Si el formato de denominación provoca que dos nombres sean idénticos, EGL añade un sufijo a cada archivo generado tras el primero. El sufijo es el siguiente:

\$vn

donde

**n** Es un entero asignado en orden secuencial, empezando por 2.

### Conceptos relacionados

“Plan de construcción” en la página 327

“Archivo de entorno J2EE” en la página 357

“Programa Java, PageHandler y biblioteca” en la página 328  
 “Archivo de propiedades de enlace” en la página 364  
 “Archivo de propiedades del programa” en la página 350  
 “Archivo de resultados” en la página 328

**Consulta relacionada**

“Elemento callLink” en la página 407

## Salida de la generación de envoltura Java

La salida de la generación de envoltura Java es la siguiente:

- Un plan de construcción, si se ha omitido la opción del descriptor de construcción **genProject**
- JavaBeans para envolver las llamadas realizadas a un programa servidor Java (consulte el apartado *Envoltura Java*)
- Beans de sesión EJB bajo determinadas circunstancias; encontrará los detalles en la explicación del elemento callLink en *Componente de opciones de enlace*
- Un archivo de resultados, si se omite **genProject**

Puede utilizar los beans generados para envolver llamadas a programas de servidor de clases Java tales como servlets, EJB o aplicaciones Java. Se generan los siguientes tipos de clases:

- Beans para servidores
- Beans para parámetros de registro
- Beans para filas de matriz de registro

La tabla siguiente muestra los nombres de los diversos tipos de componentes de envoltura Java generados:

Nombres de los componentes de envoltura Java generados

Tipo y nombre de componente	Qué se genera
Programa denominado P	Una clase denominada PWrapper en PWrapper.java
Un registro denominado R utilizado como parámetro	Una clase denominada R en R.java
Un área subestructurada S en el registro R utilizado como parámetro	Una clase denominada R.S en R.java
Componente de opciones de enlace denominado L	Archivo de propiedades de enlace denominado L.properties

1. Para los tipos de componente indicados, es posible que existan dos o más componentes con el mismo nombre. En ese caso, el nombre del segundo tendrá un sufijo adicional, \$v2. El nombre del tercero tendrá un sufijo \$v3, el cuarto tendrá \$v4, etc.

Al solicitar que un componente de programa se genere como envoltura Java, EGL produce una clase Java para cada uno de los siguientes ejecutables:

- El componente de programa
- Cada registro que se declara como un parámetro de programa
- Un bean de sesión, si especifica un componente de opciones de enlace y un elemento **callLink** para el programa generado tiene un tipo de enlace de **ejbCall**

Además, la clase generada para cada registro incluye una clase interior (o una clase dentro de una clase interna) para cada elemento de estructura que tiene estas características:

- Está en la estructura interna de uno de estos registros
- Tiene al menos un elemento de estructura subordinado; es decir, está subestructurada
- Es una matriz; en este caso, una matriz subestructurada

Cada clase generada se almacena en un archivo. El generador de EGL crea nombres utilizados en las envolturas Java como se indica a continuación:

- El nombre se convierte a minúsculas.
- Se suprime cada guió o signo menos (-) o subrayado (\_). Un carácter a continuación de un guió o subrayado se cambia a mayúsculas.
- Cuando se utiliza el nombre como un nombre de clase o dentro de un nombre de método, el primer carácter se convierte de nuevo a mayúsculas.

Si uno de los parámetros del programa es un registro, EGL genera también una clase de envoltura para esa variable. Si el programa Prog contiene un parámetro de registro con un typeDef denominado Rec, la clase de envoltura del parámetro se denominará Rec. Si el typeDef de un parámetro tiene el mismo nombre que el programa, la clase de envoltura para el parámetro tendrá un sufijo "Record".

El generador también genera una envoltura si un parámetro de registro tiene un elemento de matriz y el elemento tiene otros elementos debajo. Esta envoltura de matriz subestructurada se convierte en una clase interna de la envoltura de registro. En la mayoría de casos, un elemento de matriz de subestructura denominado AItem en Rec se envolverá en una clase denominada Rec.AItem. El registro puede contener dos elementos de matriz subestructurada con el mismo nombre, en cuyo caso las envolturas de elemento se denominan utilizando los nombres calificados de los elementos. Si el nombre calificado del primer AItem es Top1.AItem y el nombre calificado del segundo es Top2.Middle2.AItem, las clases se denominarán Rec.Top1\$\_aitem y Rec.Top2\$\_middle2\$\_aitem. Si el nombre de una matriz subestructurada es el mismo que el nombre del programa, la clase de envoltura para la matriz subestructurada tendrá un sufijo Structure.

Se generan métodos para establecer y obtener el valor de elementos de bajo nivel en cada envoltura de registro y envoltura de matriz subestructurada. Si dos elementos de bajo nivel del registro o matriz subestructurada tienen el mismo nombre, el generador utiliza el esquema de nombre calificado descrito en el párrafo anterior.

Se generan métodos adicionales en envolturas para variables de registro SQL. Para cada elemento de la variable de registro, el generador crea métodos para obtener y establecer su valor de indicador nulo y métodos para obtener y establecer su indicador de longitud SQL.

Puede utilizar la herramienta Javadoc para construir un archivo *nombreclase.html* una vez se haya compilado la clase. El archivo HTML describe las interfaces públicas para la clase. Si utiliza archivos HTML creados por Javadoc, asegúrese de que son una envoltura Java. Los archivos HTML generados desde una envoltura Java de VisualAge son distintos de los generados desde una envoltura Java de EGL.

## Ejemplo

A continuación se muestra un ejemplo de un componente de registro con una matriz subestructurada:

```
Record myRecord type basicRecord
 10 MyTopStructure[3];
 15 MyStructureItem01 CHAR(3);
 15 MyStructureItem02 CHAR(3);
end
```

En relación con el componente de programa, el archivo de salida se denomina como se indica a continuación:

*aliasWrapper.java*

donde

### **alias**

Es el nombre de alias, si existe, que se ha especificado en el componente de programa. Si no se ha especificado el nombre externo, se utiliza el nombre del componente de programa.

En relación con cada registro declarado como un parámetro de programa, el archivo de salida se denomina como se indica a continuación:

*nombreRegistro.java*

donde

### **nombreRegistro**

Es el nombre del componente de registro

En relación con una matriz subestructurada, el nombre y posición de la clase interna depende de si el nombre de matriz es exclusivo en el registro:

- Si el nombre de matriz es exclusivo en el registro, la clase interna se encuentra dentro de la clase de registro y se denomina como se indica a continuación:

*nombreRegistro.nombreIs*

donde

### **nombreRegistro**

Es el nombre del componente de registro

### **nombreIs**

Es el nombre de la matriz

- Si el nombre de matriz no es exclusivo en el registro, el nombre de la clase interna se basa en el nombre totalmente calificado de la matriz, con un calificador separado del siguiente por una combinación del signo de dólar y (\$) subrayado (\_). Por ejemplo, si la matriz está en el tercer nivel del registro, la clase generada es una clase interna de la clase de registro y se denomina como se indica a continuación:

*nombreSuperior\$\_segundoNombre\$\_nombreIs*

donde

### **nombreSuperior**

Es el nombre del elemento de estructura de nivel superior

### **segundoNombre**

Es el nombre del elemento de estructura de segundo nivel

### **nombreIs**

Es el nombre del elemento de matriz subestructurada

Si hay otra matriz con el mismo nombre subordinada al nivel superior del registro, la clase interna también estará dentro de la clase de registro y se denomina como se indica a continuación:

`nombreSuperior$_nombreIs`

donde

**nombreSuperior**

Es el nombre del elemento de estructura de nivel superior

**nombreIs**

Es el nombre del elemento de matriz subestructurada

Finalmente, observe el caso siguiente: una matriz subestructurada tiene un nombre que no es exclusivo en el registro y la matriz está subordinada a otra matriz subestructurada cuyo nombre no es exclusivo en el registro. La clase para la matriz subordinada se genera como una clase interna de una clase interna.

Al generar una envoltura Java, también puede generar un archivo de propiedades Java y un archivo de propiedades de enlace si solicita establecer opciones de enlace durante la ejecución.

**Conceptos relacionados**

“Plan de construcción” en la página 327

“Bean de sesión EJB (Enterprise JavaBean)” en la página 315

“Envoltura Java” en la página 301

“Componente de opciones de enlace” en la página 311

“Archivo de propiedades de enlace” en la página 364

“Archivo de resultados” en la página 328

**Tareas relacionadas**

“Generar envolturas Java” en la página 300

**Consulta relacionada**

“Elemento callLink” en la página 407

“Clases de envoltura Java” en la página 551

---

## Componente PageHandler en formato fuente EGL

Un componente pageHandler se declara en un archivo EGL, que está descrito en *Proyectos, paquetes y archivos EGL*. Este componente es un componente generable, lo que significa que debe estar en el nivel superior del archivo y debe tener el mismo nombre que el archivo.

Este es un ejemplo de un componente pageHandler:

```
// Page designer requiere que todos los pageHandler
// estén en un paquete llamado "pagehandlers".
package pagehandlers ;

PageHandler ListCustomers
{onPageLoadFunction="onPageLoad"}

// Biblioteca para acceso de tabla cliente
use CustomerLib3;

// Lista de clientes
customerList Customer[] {maxSize=100};

Function onPageLoad()

// Clave inicial para recuperar clientes
startkey CustomerId;
```

```

// Resultado de llamada de biblioteca
status int;

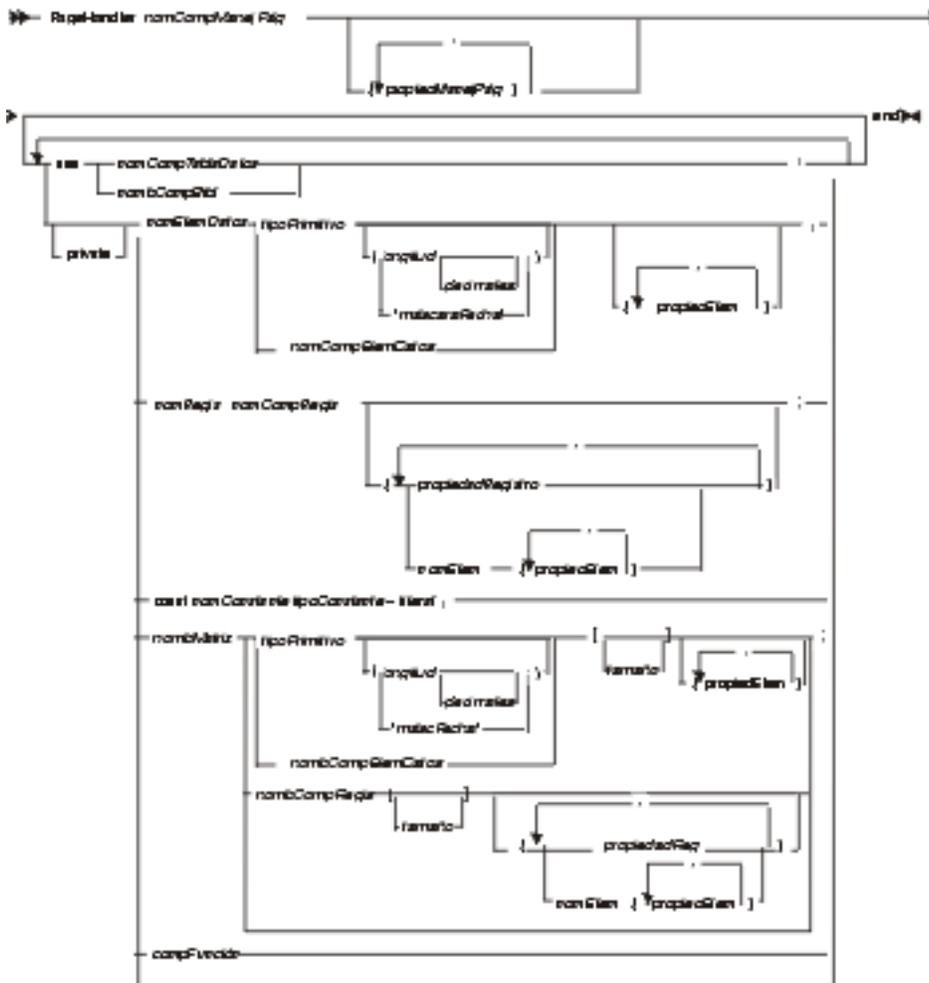
// Recuperar hasta 100 registros de cliente
startKey = 0;
CustomerLib3.getCustomersByCustomerId(startKey,
customerList, status);

if (status != 0 && status != 100)
setError("Ha fallado la recuperación de clientes.");
end
end

Function returnToIntroductionClicked()
forward to "Introducción";
end
End

```

El diagrama de un componente pageHandler es el siguiente:



**PageHandler nombreComponenteManejadorPáginas ... end**

Identifica el componente como PageHandler y especifica el nombre del componente. Para conocer las reglas de denominación, consulte *Convenios de denominación*.

*propiedadManejadorPáginas*

Una propiedad del componente PageHandler, tal como se lista en *Propiedades del componente PageHandler*.

**use** *nombreComponenteTablaDatos*, **use** *nombreComponenteBiblioteca*

Una declaración use que simplifica el acceso de una tabla de datos o biblioteca. Para conocer detalles, consulte *Declaración de uso*.

**private**

Indica que la variable, constante o función no está disponible para la JSP que muestra la página Web. Si omite el término **private**, puede enlazar la variable, constante o función a un control en la página Web.

*nombreElementoDatos*

El nombre de un elemento de datos (una variable). Para conocer las reglas, consulte *Convenios de denominación*.

*tipoPrimitivo*

El tipo primitivo asignado al elemento de datos.

*longitud*

La longitud del elemento de estructura, que es un entero. El valor de un área de memoria que se basa en el elemento de estructura incluye el número de caracteres o dígitos especificado.

*decimales*

Para un tipo numérico (BIN, DECIMAL, NUM, NUMC o PACF), puede especificar *decimales*, que es un entero que representa el número de posiciones después de la coma decimal. El número máximo de posiciones decimales es el menor de dos números: 18 o el número de dígitos declarado como *longitud*. La coma decimal no se almacena con los datos.

*nombreComponenteElementoDatos*

El nombre de un componente dataItem que es un modelo de formato para el elemento de datos, tal como se describe en *typeDef*. El componente dataItem debe ser visible al componente pageHandler, tal como se describe en *Referencias a componentes*.

*propiedadElemento*

Una propiedad de elemento. Encontrará los detalles en *Propiedades de elemento de página*.

*nombreRegistro*

El nombre de un registro (una variable). Para conocer las reglas, consulte *Convenios de denominación*.

*nombreComponenteRegistro*

El nombre de un componente de registro que es un modelo de formato para el registro, tal como se describe en *typeDef*. El componente de registro debe ser visible al componente pageHandler, tal como se describe en *Referencias a componentes*.

*propiedadRegistro*

Una alteración temporal de una propiedad de registro. Para conocer detalles sobre las propiedades de registros, consulte una de las siguientes descripciones, dependiendo del tipo de registro indicado en *nombreComponenteRegistro*:

- Componente de registro básico en formato fuente EGL
- Componente de registro indexado en formato fuente EGL
- Componente de registro MQ en formato fuente EGL
- Componente de registro relativo en formato fuente EGL

- Componente de registro en serie en formato fuente EGL
- Componente de registro SQL en formato fuente EGL

*nombreElemento*

El nombre del elemento de registro cuyas propiedades desea alterar temporalmente.

*propiedadElemento*

Una alteración temporal de una propiedad de elemento. Encontrará los detalles en *Visión general de propiedades y alteraciones temporales de EGL*.

*nombreConstante literal*

El nombre y valor de una constante. Para conocer las reglas, consulte *Convenios de denominación*.

*nombreMatriz*

El nombre de una matriz de registros o elementos de datos dinámica o estática. Si utiliza esta opción, los demás símbolos a la derecha (*nombreComponenteElementoDatos*, *tipoPrimitivo* y demás) hacen referencia a cada elemento de la matriz.

*componenteFunción*

Una función incorporada. Para conocer detalles sobre la sintaxis, consulte *Componente de función en formato fuente EGL*.

**Conceptos relacionados**

“Proyectos, paquetes y archivos EGL” en la página 13

“Visión general de las propiedades de EGL” en la página 64

“PageHandler” en la página 194

“Referencias a componentes” en la página 21

“Referencias a variables en EGL” en la página 58

“Typedef” en la página 27

**Consulta relacionada**

“Manejo de excepciones” en la página 94

“Componente de función en formato fuente EGL” en la página 527

“Convenios de denominación” en la página 672

“Propiedades del campo PageHandler” en la página 685

“Propiedades del componente PageHandler”

“Tipos primitivos” en la página 34

“Compatibilidad de referencia en EGL” en la página 739

“setError()” en la página 906

“Declaración use” en la página 954

## Propiedades del componente PageHandler

Si se excluyen las propiedades que son específicas de los *campos* de PageHandler, las propiedades del PageHandler son las siguientes y son opcionales:

**alias = "alias"**

Una serie que se incorpora a los nombres de la salida generada. Si no especifica un alias, se utilizará en su lugar el nombre del componente PageHandler.

**allowUnqualifiedItemReferences = no, allowUnqualifiedItemReferences = yes**

Especifica si debe permitirse que el código haga referencia a elementos de estructura, pero excluyendo el nombre del *contenedor*, que es la tabla de datos, registro o formulario que contiene el elemento de estructura. Observe, por ejemplo, el siguiente componente de registro:

```
Record aRecordPart type basicRecord
 10 myItem01 CHAR(5);
 10 myItem02 CHAR(5);
end
```

La variable siguiente se basa en ese componente:

```
myRecord aRecordPart;
```

Si acepta el valor por omisión de **allowUnqualifiedItemReferences** (*no*), debe especificar el nombre de registro al hacer referencia a `myItem01`, como en esta asignación:

```
myValue = myRecord.myItem01;
```

Sin embargo, si establece la propiedad **allowUnqualifiedItemReferences** en *yes*, puede evitar especificar el nombre de registro:

```
myValue = myItem01;
```

Se recomienda aceptar el valor predeterminado, que ofrece un mejor resultado. Al especificar el nombre de contenedor, se reduce la ambigüedad para quien lea su código y para EGL.

EGL utiliza un conjunto de reglas para determinar el área de la memoria a la que hace referencia un nombre de variable o un nombre de elemento. Para obtener detalles, consulte el apartado *Referencias a variables y constantes*.

#### **handleHardIOErrors = yes, handleHardIOErrors = no**

Establece el valor predeterminado para la variable del sistema

**VGVar.handleHardIOErrors**. La variable controla si un programa continúa ejecutándose después de que se haya producido un error grave en una operación de E/S en un bloque `try`. El valor predeterminado de la propiedad es *yes*, que establece la variable en 1.

Para obtener más detalles, consulte *VGVar.handleHardIOErrors* y *Manejo de excepciones*.

#### **includeReferencedFunctions = no, includeReferencedFunctions = yes**

Indica si el bean de `PageHandler` contiene una copia de cada función que no está ni dentro del `PageHandler` ni en una biblioteca a la que éste accede. El valor predeterminado es *no*, lo que significa que puede ignorar esta propiedad si cumple las siguientes prácticas en el momento del desarrollo, tal como se recomienda:

- Colocar funciones compartidas en una biblioteca
- Colocar funciones no compartidas en el `PageHandler`

Si utiliza funciones compartidas que no están en una biblioteca, la generación sólo es posible si establece la propiedad **includeReferencedFunctions** en *yes*.

#### **localSQLScope = no, localSQLScope = yes**

Indica si los identificadores para los conjuntos de resultados SQL y las sentencias preparadas son locales para el `pageHandler`, lo que constituye el valor predeterminado. Si acepta el valor *yes*, los distintos programas llamados por el `pageHandler` pueden utilizar independientemente los mismos identificadores.

Si especifica *no*, los identificadores se comparten en toda la unidad de ejecución. Los identificadores creados en el código actual están disponibles en cualquier parte, aunque otro código puede utilizar **localSQLScope = yes** para bloquear el acceso a esos identificadores. Además, el código actual puede hacer

referencia a identificadores creados en cualquier parte, pero solo si el otro código ya se ha ejecutado y no ha bloqueado el acceso.

Los efectos de compartir identificadores SQL son los siguientes:

- Puede abrir un conjunto de resultados en un `pageHandler` o programa llamado y obtener filas de ese conjunto en el otro código
- Puede preparar una sentencia SQL en un código y ejecutar esa sentencia en otra

**`msgResource = "nombreLógico"`**

Identifica un empaquetamiento de recursos o archivo de propiedades Java utilizado en la presentación de mensajes de error. El contenido del empaquetamiento de recursos o archivo de propiedades se compone de un conjunto de claves y los valores relacionados.

Se visualiza un valor determinado en respuesta a la invocación de la función de sistema EGL `sysLib.setError` por parte del programa, cuando la invocación incluye la utilización de la clave correspondiente a ese valor.

**`onPageLoadFunction = "nombreFunción"`**

El nombre de una función de `PageHandler` que recibe el control cuando el JSP relacionado visualiza inicialmente una página Web. La función puede utilizarse para configurar los valores iniciales de los datos visualizados en la página. Esta propiedad era anteriormente la propiedad **`onPageLoad`**.

Los argumentos que se pasan a la función deben tener compatibilidad de referencia, como se describe en el apartado *Compatibilidad de referencia en EGL*.

**`scope = session, scope = request`**

Especifica lo que ocurre después de enviar los datos de `pageHandler` a la página Web:

- Si el ámbito se establece en *session* (lo que constituye el valor predeterminado), los valores de la variable `pageHandler` se retienen en la sesión de usuario y el último acceso de usuario del mismo `pageHandler` no vuelve a invocar la función `OnPageLoad`
- Si el ámbito se establece en *request*, los valores de la variable `pageHandler` se pierden y el acceso de usuario del mismo `PageHandler` vuelve a invocar la función `OnPageLoad`

Es recomendable establecer esta propiedad explícitamente para documentar la decisión lo que afecta mucho al diseño y el funcionamiento de la aplicación Web.

**`throwNrfEofExceptions = no, throwNrfEofExceptions = yes`**

Especifica si un error leve provoca el lanzamiento de una excepción. El valor predeterminado es *no*. Para obtener información, consulte la sección *Manejo de excepciones*.

**`title = "literal"`**

La propiedad **`title`** es una propiedad de enlace, lo que significa que el valor asignado se utiliza como valor por omisión al trabajar en Page Designer. La propiedad especifica el título de la página.

*literal* es una serie entrecomillada.

**`validationBypassFunctions = ["nombresFunción"]`**

Identifica uno o varios *manejadores de eventos*, que son funciones de `PageHandler` asociadas con un control de botón en el JSP. Cada nombre de función está separado del siguiente por una coma.

Si especifica un manejador de eventos en este contexto, el entorno de ejecución de EGL se salta las validaciones de campos de entrada y páginas cuando el usuario pulsa el botón o en el enlace de hipertexto relacionado con el manejador de eventos. Esta propiedad resulta de utilidad para reservar una acción de usuario que finaliza el proceso del PageHandler actual y que transfiere inmediatamente el control a otro recurso Web.

**validatorFunction** = "*nombreFunción*"

Identifica la función de validador de PageHandler, que se invoca tras invocar todos los validadores de elementos, tal como se describe en *Validación en aplicaciones Web creadas con EGL*. Esta propiedad era anteriormente la propiedad **validator**.

**view** = "*nombreArchivoJSP*"

Identifica el nombre y la vía de acceso al subdirectorio del JSP (Java Server Page) que está enlazado al PageHandler. *nombreArchivoJSP* es una serie entrecomillada.

El valor predeterminado es el nombre del PageHandler, con la extensión de archivo **.jsp**. Si especifica esta propiedad, incluya la extensión de archivo, si la hay.

Al guardar o generar un PageHandler, EGL añade un archivo JSP al proyecto para la personalización posterior, a menos que un archivo JSP con el mismo nombre (el nombre especificado en la propiedad **view**) ya se encuentre en la carpeta adecuada (la carpeta WebContent\WEB-INF). EGL nunca altera temporalmente un JSP.

## Propiedades del campo PageHandler

Las propiedades del campo PageHandler especifican características que son relevantes cuando un campo se declara en un componente PageHandler.

Las propiedades son las siguientes:

- "action" en la página 690
- "byPassValidation" en la página 691
- "displayName" en la página 697
- "displayUse" en la página 698
- "help" en la página 699
- "newWindow" en la página 707
- "numElementsItem" en la página 708
- "selectFromListItem" en la página 711
- "selectType" en la página 712
- "validationOrder" en la página 717
- "value" en la página 722

### Conceptos relacionados

"Visión general de las propiedades de EGL" en la página 64

"PageHandler" en la página 194

### Tareas relacionadas

"Crear un componente EGL pageHandler" en la página 191

"Utilizar la vista Edición rápida para el código de PageHandler" en la página 202

### Consulta relacionada

"Propiedades del componente PageHandler" en la página 682

“Componente PageHandler en formato fuente EGL” en la página 679  
“Soporte de Page Designer para EGL” en la página 192

---

## pfKeyEquate

Al declarar un grupo de formularios que hace referencia a un formulario de texto, la propiedad *pfKeyEquate* especifica si la pulsación que se registra cuando el usuario pulsa una tecla de función con un número alto (de PF13 a PF24) es la misma que la pulsación registrada cuando el usuario pulsa una tecla de función inferior a 12.

Si acepta el valor por omisión *yes* para *pfKeyEquate*, las expresiones lógicas sólo pueden hacer referencia a 12 de las teclas de función, ya que (por ejemplo) PF2 es lo mismo que PF14.

**Nota:** Las teclas de función de un teclado de PC son con frecuencia teclas *F*, como por ejemplo F1, pero EGL utiliza la terminología IBM *PF* a fin de que (por ejemplo) F1 se denomine PF1.

### Conceptos relacionados

“Componente FormGroup” en la página 153

### Consulta relacionada

“Componente FormGroup en formato fuente EGL” en la página 508

---

## Propiedades a nivel de campo primitivo

La tabla siguiente lista las propiedades a nivel de campo primitivo en EGL:

Propiedad	Descripción
action	Identifica el código que se invoca cuando el usuario pulsa el botón o el enlace.
align	Especifica la posición de datos en un campo de variable cuando la longitud de los datos es menor que la longitud del campo.
byPassValidation	Identifica si se pasa por alto la validación basada en EGL cuando el usuario pulsa el botón o el enlace.
color	Especifica el color de un campo de un formulario de texto.
column	Hace referencia al nombre de la columna de tabla de base de datos que está asociada con el elemento. El valor por omisión es el nombre del elemento.
currency	Indica si debe incluirse un símbolo de moneda antes del valor en un campo numérico, con la posición exacta del símbolo determinada por la propiedad <b>zeroFormat</b> .
currencySymbol	Indica qué símbolo de moneda se debe utilizar cuando la propiedad <b>currency</b> está en vigor.
dateFormat	Identifica el formato para las fechas.

Propiedad	Descripción
	Especifica si se establece el código de datos modificados del campo cuando éste se selecciona mediante un lápiz óptico o (en sesiones de emulador) mediante una pulsación del cursor.
displayName	Especifica la etiqueta que se visualiza junto al campo.
displayUse	Asocia un campo de EGL con un control de interfaz de usuario.
fieldLen	Especifica el número de caracteres de un solo byte que pueden visualizarse en un campo de formulario de texto.
fill	Indica si es necesario que el usuario especifique datos en cada posición de campo.
fillCharacter	Indica qué carácter rellena posiciones no utilizadas en un formulario de texto o impresión o en datos de manejador de páginas.
help	Especifica el texto de ayuda flotante que se visualiza cuando el usuario sitúa el cursor sobre el campo de entrada.
highlight	Especifica el efecto especial (si lo hay) con el que se visualizará el campo.
inputRequired	Indica si es necesario que el usuario especifique datos en el campo.
inputRequiredMsgKey	Identifica el mensaje que se visualiza si la propiedad de campo <b>inputRequired</b> se establece como <i>sí</i> y el usuario no coloca datos en el campo.
intensity	Especifica la fuerza del font visualizado.
isBoolean	Indica que el campo representa un valor booleano.
isDecimalDigit	Determina si debe comprobarse si el valor de entrada incluye solamente dígitos decimales
isHexDigit	Determina si debe comprobarse si el valor de entrada incluye solamente dígitos hexadecimales
isNullable	Indica si el elemento puede establecerse en nulo, como es adecuado si la columna de tabla asociada con el elemento puede establecerse en NULL.
isReadOnly	Indica si el elemento y la columna relacionada deben omitirse de las sentencias SQL por omisión que escriben en la base de datos o incluyen una cláusula FOR UPDATE OF.
lineWrap	Indica si el texto puede acomodarse en una línea nueva, si ello es necesario para no truncar el texto.

Propiedad	Descripción
lowerCase	Indica si deben establecerse caracteres alfabéticos en minúsculas en la entrada de caracteres de un solo byte del usuario.
masked	Indica si se visualizará el carácter entrado por el usuario.
maxLen	Especifica la longitud máxima de texto de campo que se escribe en la columna de base de datos.
minimumInput	Indica el número mínimo de caracteres que el usuario debe especificar en el campo, si el usuario especifica datos en el campo.
minimumInputMsgKey	Identifica el mensaje que se visualiza si el usuario actúa como se indica a continuación: <ul style="list-style-type: none"> <li>• Especifica datos en el campo; y</li> <li>• Especifica menos caracteres que el valor especificado en la propiedad <b>minimumInputRequired</b>.</li> </ul>
modified	Indica si el programa considerará el campo como modificado, independientemente de que el usuario haya cambiado el valor.
needsSOSI	Indica si EGL realiza una comprobación especial cuando el usuario especifica datos de tipo MBCHAR en un dispositivo ASCII.
newWindow	Indica si debe utilizarse una ventana de navegador nueva cuando el entorno de ejecución EGL presenta una página Web en respuesta a la actividad identificada en la propiedad <b>action</b> .
numElementsItem	Identifica un campo de PageHandler cuyo valor de tiempo de ejecución especifica el número de elementos de matriz a visualizar.
numericSeparator	Indica si debe colocarse un carácter en un número que tenga un componente entero de más de 3 dígitos.
outline	Permite trazar líneas en los bordes de campos de cualquier dispositivo que dé soporte a caracteres de doble byte.
pattern	Empareja el texto introducido por el usuario respecto a un patrón especificado, a efectos de validación.
persistent	Indica si el campo se incluye en las sentencias SQL implícitas generadas para el registro SQL.
protect	Especifica si el usuario puede acceder al campo.
selectFromListItem	Identifica la matriz o la columna de DataTable desde la que el usuario selecciona un valor o varios valores, los cuales se transfieren a la matriz o campo primitivo que se está declarando.

Propiedad	Descripción
selectType	Indica la clase de valor que se recupera en la matriz o campo primitivo que se declara.
sign	Indica la posición en la que se visualiza un signo positivo (+) o negativo (-) cuando se coloca un número en el campo, ya sea desde entrada de usuario o desde el programa.
sqlDataCode	Identifica el tipo de datos SQL que están asociados con el elemento de registro.
sqlVariableLen	Indica si los blancos finales y los nulos de un campo de caracteres se truncan antes de que el entorno de ejecución EGL escriba los datos en una base de datos SQL.
timeFormat	Identifica el formato para las horas.
timeStampFormat	Identifica el formato para indicaciones de la hora que se visualizan en un formulario o que se mantienen en un PageHandler.
typeChkMsgKey	Identifica el mensaje que se visualiza si los datos de entrada no son adecuados para el tipo de campo.
upperCase	Indica si deben establecerse caracteres alfabéticos en mayúsculas en la entrada de caracteres de un solo byte del usuario.
validationOrder	Indica cuándo se ejecuta la función de validador del campo en relación con la función de validador de cualquier otro campo.
validatorDataTable	Identifica una <i>tabla de validación</i> , que es un componente dataTable que actúa como base de una comparación con la entrada del usuario.
validatorDataTableMsgKey	Identifica el mensaje que se visualiza si el usuario proporciona datos que no corresponden a los requisitos de la <i>tabla de validación</i> , que es la tabla especificada en la propiedad <b>validatorDataTable</b> .
validatorFunction	Identifica una función de validación, que es la lógica que se ejecuta después de que el entorno de ejecución de EGL realice las comprobaciones de validación elementales, si las hay.
validatorFunctionMsgKey	Identifica un mensaje que se visualiza
validValues	Indica un conjunto de valores que son válidos para la entrada de usuario.
validValuesMsgKey	Identifica el mensaje que se visualiza si se establece la propiedad de campo <b>validValues</b> y el usuario coloca datos fuera de rango en el campo.
value	Identifica un literal de tipo serie que se visualiza como contenido del campo cuando se visualiza una página Web.

Propiedad	Descripción
zeroFormat	Especifica cómo se visualizan los valores cero en los campos numéricos, pero no en campos de tipo MONEY.

## action

Cuando la propiedad de EGL **displayUse** es *button* o *hyperlink*, la propiedad **action** identifica el código que se invoca cuando el usuario pulsa el botón o el enlace. El valor que se asigna a **action** se utiliza como valor predeterminado cuando se coloca el campo (o un registro que incluye el campo) en la página Web de Page Designer.

El valor de **action** es uno de los siguientes tipos de literales de serie:

- El nombre de una función de manejo de eventos del PageHandler
- Una etiqueta que efectúa una correlación con un recurso Web (por ejemplo, un JSP) y que corresponde a un atributo from-outcome de una entrada navigation-rule del archivo de Recursos de configuración de aplicación de JSF
- El nombre de un método de un bean Java, en cuyo caso se aplican estas normas:
  - El formato es el nombre del bean seguido de un punto y de un nombre de método
  - El nombre de bean debe estar relacionado con una de las entradas de nombre de bean gestionado del archivo de Recursos de configuración de aplicación de JSF

Si no especifica un valor para **action**, la pulsación del campo por parte del usuario tiene el efecto siguiente:

- Si el valor de la propiedad **displayUse** es *button*, se produce una validación, después de lo cual JSF vuelve a visualizar la misma página Web.
- Si el valor de la propiedad **displayUse** es *hyperlink*, se produce ninguna validación, pero JSF vuelve a visualizar la misma página Web.

### Conceptos relacionados

“Visión general de las propiedades de EGL” en la página 64  
 “PageHandler” en la página 194

### Tareas relacionadas

“Enlazar componente mandato JavaServer Faces con PageHandler EGL” en la página 201  
 “Crear un componente EGL pageHandler” en la página 191  
 “Utilizar la vista Edición rápida para el código de PageHandler” en la página 202

### Consulta relacionada

“Propiedades del campo PageHandler” en la página 685  
 “Propiedades del componente PageHandler” en la página 682  
 “Componente PageHandler en formato fuente EGL” en la página 679  
 “Soporte de Page Designer para EGL” en la página 192

## align

La propiedad de alineación, **align**, especifica la posición de datos en un campo de variable cuando la longitud de los datos es menor que la longitud del campo.

Los valores son de la enumeración **alignKind**:

**left**

Colocar los datos a la izquierda del campo, que es el valor por omisión para los datos de tipo carácter. Los espacios iniciales se colocan al final del campo.

**none**

No justificar los datos. Este valor solamente es válido para los datos de tipo carácter.

**right**

Colocar los datos a la derecha del campo, que es el valor por omisión para los datos de tipo numérico. Los espacios finales se colocan al principio del campo. Este valor es necesario para los datos numéricos que tengan un signo o posición decimal.

La propiedad está disponible en los componentes `DataItem` y es relevante para los campos que aparecen en los siguientes contextos:

- Formularios de consola
- Formularios de impresión
- Formularios de texto
- Páginas Web

En la salida, los datos de tipo carácter y numérico resultan afectados por esta propiedad. En la entrada, los datos de tipo carácter resultan afectados por esta propiedad, pero los datos de tipo numérico siempre quedan afectados por esta propiedad, pero los datos de tipo numérico siempre quedan alineados a la derecha.

**Conceptos relacionados**

“Enumeraciones en EGL” en la página 484

“Visión general de las propiedades de EGL” en la página 64

**Consulta relacionada**

“Propiedades de formato” en la página 66

## byPassValidation

Cuando la propiedad de EGL **displayUse** es *button* o *hyperlink*, la propiedad **byPassValidation** indica si se salta la validación basada en EGL cuando el usuario pulsa el botón o el enlace. Es posible que desee saltarse la validación para obtener un mejor rendimiento; por ejemplo, siempre que el usuario pulsa un botón Salir.

El valor que se asigna a **byPassValidation** se utiliza como valor predeterminado cuando se coloca el campo (o un registro que incluye el campo) en la página Web de Page Designer.

La propiedad sólo afecta a las validaciones basadas en EGL y no a las especificadas por los códigos JSF; para obtener información detallada, consulte el apartado *PageHandler*.

Los valores son los de la enumeración **Boolean**:

**no (valor predeterminado)**

Los campos de entrada se validan como siempre

**yes**

El entorno de ejecución EGL no devuelve los datos de usuario al *PageHandler*

### Conceptos relacionados

“Enumeraciones en EGL” en la página 484

“Visión general de las propiedades de EGL” en la página 64

“PageHandler” en la página 194

### Tareas relacionadas

“Enlazar componente mandato JavaServer Faces con PageHandler EGL” en la página 201

“Crear un componente EGL pageHandler” en la página 191

“Utilizar la vista Edición rápida para el código de PageHandler” en la página 202

### Consulta relacionada

“Propiedades del campo PageHandler” en la página 685

“Propiedades del componente PageHandler” en la página 682

“Componente PageHandler en formato fuente EGL” en la página 679

“Soporte de Page Designer para EGL” en la página 192

## color

La propiedad **color** especifica el color de un campo en un formulario de texto. Puede seleccionar cualquiera de los siguientes:

- negro
- azul
- cian
- defaultColor (el valor predeterminado)
- verde
- magenta
- rojo
- blanco
- amarillo

Si asigna el valor *defaultColor*, otras condiciones determinarán el color visualizado, como se muestra en la tabla siguiente.

¿Se asigna el valor <i>defaultColor</i> a todos los campos del formulario?	Valor de <i>protect</i>	Valor de <i>intensity</i>	Color visualizado para un campo al que se ha asignado el valor <i>defaultColor</i>
yes	<i>yes</i> o <i>skip</i>	no <i>bold</i>	azul
yes	<i>yes</i> o <i>skip</i>	<i>bold</i>	blanco
yes	<i>no</i>	no <i>bold</i>	verde
yes	<i>no</i>	<i>bold</i>	rojo
no	cualquier valor	no <i>bold</i>	verde
no	cualquier valor	<i>bold</i>	blanco

### Conceptos relacionados

“Enumeraciones en EGL” en la página 484

“Visión general de las propiedades de EGL” en la página 64

### Consulta relacionada

“Propiedades de presentación de campos” en la página 66

## column

La propiedad **column** hace referencia al nombre de la columna de tabla de base de datos que está asociada con el elemento. El valor por omisión es el nombre del elemento. La columna y el elemento relacionado afectan a las sentencias SQL por omisión, como se describe en el apartado *Soporte de SQL*.

Sustituya "*nombreColumna*" por una serie entrecomillada, una variable de tipo carácter o una concatenación, como en este ejemplo:

```
column = "Columna" + "01"
```

Si un nombre de columna es una de las siguientes palabras reservadas SQL, se aplica una sintaxis especial:

- CALL
- COLUMNS
- FROM
- GROUP
- HAVING
- INSERT
- ORDER
- SELECT
- SET
- UPDATE
- VALUES
- WHERE

Como se muestra en el ejemplo siguiente, cada uno de esos nombres debe especificarse entre comillas y cada una de las comillas debe ir precedida de un carácter de escape (\):

```
column = "\"SELECT\""
```

(Una situación similar se aplica si utiliza estas palabras reservadas como nombres de tabla).

### Conceptos relacionados

"Compatibilidad con VisualAge Generator" en la página 439

"Tipos de registros y propiedades" en la página 135

"Soporte de SQL" en la página 229

"Estructura fija" en la página 26

"Typedef" en la página 27

### Tareas relacionadas

"Recuperar datos de tabla SQL" en la página 252

### Consulta relacionada

"Propiedades de presentación de campos" en la página 66

"add" en la página 561

"close" en la página 568

"Inicialización de datos" en la página 471

"delete" en la página 571

"execute" en la página 574

"get" en la página 585

"get next" en la página 597

“open” en la página 616  
“prepare” en la página 630  
“Tipos primitivos” en la página 34  
“Referencias cruzadas de tipo de registro y tipo de archivo” en la página 737  
“replace” en la página 632  
“set” en la página 636  
“Códigos de datos SQL y variables de lenguaje principal EGL” en la página 744  
“terminalID” en la página 938  
“VAGCompatibility” en la página 405

## currency

La propiedad de moneda, **currency**, indica si debe incluirse un símbolo de moneda antes del valor en un campo numérico, con la posición exacta del símbolo determinada por la propiedad **zeroFormat**. El formateo de los campos de tipo MONEY depende del valor de **strLib.defaultMoneyFormat** y no se ve afectado por la propiedad **currency**.

Los valores de la propiedad **currency** son los siguientes:

### No (el valor por omisión)

No utilizar un símbolo de moneda.

### Yes

Utilice el símbolo especificado en **currencySymbol**. Si aquí no se especifica ningún valor, utilice el símbolo de la moneda por omisión.

El símbolo de moneda por omisión viene determinado por el entorno local del sistema.

La propiedad está disponible en los componentes DataItem y es relevante para los campos que aparecen en los siguientes contextos:

- Formularios de impresión
- Formularios de texto
- Páginas Web

Esta propiedad se utiliza en la entrada y en la salida.

### Conceptos relacionados

“Enumeraciones en EGL” en la página 484

“Visión general de las propiedades de EGL” en la página 64

### Consulta relacionada

“Propiedades de formato” en la página 66

## currencySymbol

La propiedad **currencySymbol** indica qué símbolo de moneda se debe utilizar cuando la propiedad **currency** está en vigor. El valor es un literal de tipo serie.

La propiedad está disponible en los componentes DataItem y es relevante para los campos que aparecen en los siguientes contextos:

- Formularios de impresión
- Formularios de texto
- Páginas Web

Esta propiedad se utiliza en la entrada y en la salida.

### Conceptos relacionados

"Enumeraciones en EGL" en la página 484

"Visión general de las propiedades de EGL" en la página 64

### Consulta relacionada

"Propiedades de formato" en la página 66

## dateFormat

La propiedad **dateFormat** identifica el formato de las fechas.

Los valores válidos son los siguientes:

### "patrón"

El valor de *patrón* consiste en un conjunto de caracteres, tal como se describe en la sección *Especificadores de formato de fecha, hora e indicación de la hora*.

Los caracteres pueden eliminarse desde el inicio o el final de una especificación de fecha completa, pero no desde el medio.

### defaultDateFormat

Si se especifica para un campo de página, el valor de **defaultDateFormat** es el formato de fecha dado en el entorno local de ejecución de Java. Si se especifica para un campo de formulario, el patrón predeterminado es equivalente a seleccionar **systemGregorianCalendar**.

### eurDateFormat

El patrón "dd.MM.aaaa", que es el formato de fecha estándar europeo de IBM.

### isoDateFormat

El patrón "aaaa-MM-dd", que es el formato de fecha especificado por International Standards Organization (ISO).

### jisDateFormat

El patrón "aaaa-MM-dd" que es el formato de fecha estándar industrial japonés.

### usaDateFormat

El patrón "MM/dd/aaaa", que es el formato de fecha estándar para EE.UU. de IBM.

### systemGregorianCalendar

Un patrón de 8 ó 10 caracteres que incluye dd (para día numérico), MM (para mes numérico) y aa o aaaa (para año numérico), con caracteres que no sean d, M, a ni dígitos utilizados como separadores.

El formato está en la siguiente propiedad de entorno de ejecución Java:

```
vgj.datemask.gregorian.long.NLS
```

### NLS

El código NLS (soporte de idioma nacional) especificado en la propiedad de entorno de ejecución Java **vgj.nls.code**. El código es uno de los que se listan en targetNLS. Inglés en mayúsculas (código ENP) no está soportado.

Para obtener más detalles acerca de **vgj.nls.code**, consulte el apartado *Propiedades de entorno de ejecución Java (detalles)*.

### systemJulianDateFormat

Un patrón de 6 u 8 caracteres que incluye DDD (para día numérico) y aa o aaaa (para año numérico), con caracteres que no sean D, y ni dígitos como separadores.

El formato está en la siguiente propiedad de entorno de ejecución Java:

vgj.datemask.julian.long.NLS

### NLS

El código NLS (soporte de idioma nacional) especificado en la propiedad de entorno de ejecución Java **vgj.nls.code**. El código es uno de los que se listan en targetNLS. Inglés en mayúsculas (código ENP) no está soportado.

Para obtener más detalles acerca de **vgj.nls.code**, consulte el apartado *Propiedades de entorno de ejecución Java (detalles)*.

La propiedad está disponible en los componentes DataItem y es relevante para los campos que aparecen en los siguientes contextos:

- Formularios de consola
- Formularios de impresión
- Formularios de texto
- Páginas Web

Esta propiedad se utiliza para la entrada y la salida, pero no en los siguientes casos:

- El campo tiene posiciones decimales, un símbolo de moneda, un separador numérico o un signo; o bien
- El campo es de tipo DBCHAR, MBCHAR o HEX; o bien
- El campo no tiene la longitud suficiente para contener un valor que refleje la máscara. Para obtener información detallada, consulte el apartado "Consideraciones sobre longitud para fechas".

## Formatos de fecha internos

Cuando el usuario entra datos válidos, la fecha se convierte del formato especificado para el campo a un formato interno que se utiliza para la validación subsiguiente.

El formato interno para una fecha de caracteres es el mismo que el formato por omisión del sistema e incluye caracteres separadores.

Para una fecha numérica, los formatos internos son los siguientes:

- Para una fecha abreviada Gregoriana, 00aaMMdd
- Para una fecha larga Gregoriana, 00aaaaMMdd
- Para una fecha abreviada Juliana, 0aaDDD
- Para una fecha larga Juliana, 0aaaaDDD

## Consideraciones sobre longitud para fechas

En un formulario, la longitud de campo del formulario debe ser mayor o igual a la longitud de la máscara de campo que especifique. La longitud del campo debe ser suficiente para contener el formato interno de la fecha.

En un campo de página, las reglas son las siguientes:

- La longitud del campo debe ser suficiente para la máscara de fecha que especifique, pero puede ser más larga
- En el caso de un campo numérico, los caracteres separadores se excluyen del cálculo de la longitud.

En la siguiente tabla encontrará ejemplos.

Tipo de formato	Ejemplo	Longitud del campo de formulario	Longitud mínima del campo de página (tipo carácter)	Longitud válida del campo de página (tipo numérico)
Gregoriano abreviado	aa/MM/dd	8	8	6
Gregoriano largo	aaaa/MM/dd	10	10	8
Juliano abreviado	DDD-aa	6	6	5
Juliano largo	DDD-aaaa	8	8	7

### Consideraciones de E/S para fechas

Los datos entrados en un campo de variable se comprueban para asegurarse de que la fecha se ha entrado en el formato especificado. El usuario no tiene que entrar los ceros iniciales para días y meses, sino que puede especificar (por ejemplo) 8/5/1996 en lugar de 08/05/1996. El usuario que omite los caracteres separadores, no obstante, debe entrar todos los ceros iniciales.

#### Conceptos relacionados

“Propiedades de tiempo de ejecución Java” en la página 347  
“Visión general de las propiedades de EGL” en la página 64

#### Consulta relacionada

“Especificadores de fecha, hora e indicación de la hora” en la página 45  
“Propiedades de formato” en la página 66  
“Propiedades de ejecución de Java (detalles)” en la página 540

## displayName

La propiedad **displayName** especifica la etiqueta que se visualiza junto al campo. El valor que se asigna se utiliza como valor predeterminado cuando se coloca el campo (o un registro que incluye el campo) en la página Web de Page Designer.

El valor de esta propiedad es un literal de tipo serie.

#### Conceptos relacionados

“Visión general de las propiedades de EGL” en la página 64  
“PageHandler” en la página 194

#### Tareas relacionadas

“Asociar un registro EGL con un JSP Faces” en la página 200  
“Crear un componente EGL pageHandler” en la página 191  
“Crear un campo de EGL y asociarlo con un JSP Faces” en la página 199  
“Utilizar la vista Edición rápida para el código de PageHandler” en la página 202

#### Consulta relacionada

“Propiedades del campo PageHandler” en la página 685  
“Propiedades del componente PageHandler” en la página 682  
“Componente PageHandler en formato fuente EGL” en la página 679  
“Soporte de Page Designer para EGL” en la página 192

## displayUse

La propiedad **displayUse** asocia un campo de EGL con un control de interfaz de usuario. El valor que se asigna se utiliza como valor predeterminado cuando se coloca el campo (o un registro que incluye el campo) en la página Web de Page Designer.

Los valores son de la enumeración **displayUseKind**:

### **button**

El control tiene un código de mandato de botón

### **secret**

Los datos no son visibles al usuario. Este valor es adecuado para las contraseñas.

### **hyperlink**

Si la propiedad **action** es el nombre de una función de manejo de eventos, el control tiene un código de mandato de hiperenlace. Si la propiedad **action** es una etiqueta, el control tiene un código de enlace. Cuando el usuario pulsa en el enlace en cualquiera de los dos casos, no se produce validación y no se devuelven datos de entrada.

### **entrada**

El control acepta la entrada del usuario. Inicialmente, el control puede mostrar un valor que proporciona el PageHandler.

### **table**

Los datos están dentro de un código de tabla.

### **output**

La salida de campo de PageHandler, si existe, es visible en el control.

### **Conceptos relacionados**

“Enumeraciones en EGL” en la página 484

“Visión general de las propiedades de EGL” en la página 64

“PageHandler” en la página 194

### **Tareas relacionadas**

“Asociar un registro EGL con un JSP Faces” en la página 200

“Enlazar componente mandato JavaServer Faces con PageHandler EGL” en la página 201

“Crear un componente EGL pageHandler” en la página 191

“Crear un campo de EGL y asociarlo con un JSP Faces” en la página 199

“Utilizar la vista Edición rápida para el código de PageHandler” en la página 202

### **Consulta relacionada**

“Propiedades del campo PageHandler” en la página 685

“Propiedades del componente PageHandler” en la página 682

“Componente PageHandler en formato fuente EGL” en la página 679

“Soporte de Page Designer para EGL” en la página 192

## fieldLen

La propiedad **fieldLen** especifica el número de caracteres de un solo byte que pueden visualizarse en el campo de formulario de texto. Este valor no incluye el byte de atributo que lo precede.

El valor de **fieldLen** para campos numéricos debe ser lo suficientemente grande para visualizar el número mayor que el campo pueda contener, además de una coma decimal (si el número tiene posiciones decimales). El valor de **fieldLen** para

un campo de tipo CHAR, DBCHAR, MBCHAR o UNICODE debe ser lo suficientemente grande para contener los caracteres de doble byte, así como los caracteres de desplazamiento a teclado ideográfico y a teclado estándar.

El valor por omisión de **fieldLen** es el número de bytes necesario para visualizar el mayor número posible para el tipo primitivo, incluidos todos los caracteres de formato.

**Conceptos relacionados**

“Visión general de las propiedades de EGL” en la página 64

**Consulta relacionada**

“Componente de formulario en formato fuente EGL” en la página 511

## fill

La propiedad **fill** indica si es necesario que el usuario especifique datos en cada posición de campo. Los valores válidos son *no* (el valor por omisión) y *sí*.

**Conceptos relacionados**

“Formularios de texto” en la página 158

**Consulta relacionada**

“Propiedades de validación” en la página 67

“validationFailed()” en la página 791

“Componente DataTable en formato fuente EGL” en la página 473

“verifyChkDigitMod10()” en la página 913

“verifyChkDigitMod11()” en la página 914

## fillCharacter

La propiedad **fillCharacter** indica qué carácter rellena posiciones no utilizadas en un formulario de texto o impresión o en datos de PageHandler. Además, la propiedad cambia el efecto de *set field full*, tal como se describe en *set*. El efecto de esta propiedad solamente se produce en la salida.

El valor por omisión es un espacio para números y un 0 para elementos hex. El valor por omisión para tipos carácter depende del medio:

- En formularios de texto o impresión, el valor predeterminado es una serie vacía
- Para los datos de PageHandler, el valor predeterminado es blanco para los datos de tipo CHAR o MBCHAR

En los PageHandlers, el valor de **fillCharacter** debe ser un espacio (al igual que el valor predeterminado) para los elementos de tipo DBCHAR o UNICODE.

## help

La propiedad **help** especifica el texto de ayuda flotante que se visualiza cuando el usuario sitúa el cursor sobre el campo de entrada. El valor que se asigna se utiliza como valor predeterminado cuando se coloca el campo de EGL (o un registro que incluye el campo de EGL) en la página Web de Page Designer.

El valor de esta propiedad es un literal de tipo serie.

**Conceptos relacionados**

“Visión general de las propiedades de EGL” en la página 64

“PageHandler” en la página 194

### Tareas relacionadas

- “Asociar un registro EGL con un JSP Faces” en la página 200
- “Crear un componente EGL pageHandler” en la página 191
- “Crear un campo de EGL y asociarlo con un JSP Faces” en la página 199
- “Utilizar la vista Edición rápida para el código de PageHandler” en la página 202

### Consulta relacionada

- “Propiedades del campo PageHandler” en la página 685
- “Propiedades del componente PageHandler” en la página 682
- “Componente PageHandler en formato fuente EGL” en la página 679
- “Soporte de Page Designer para EGL” en la página 192

## highlight

La propiedad de resaltado, **highlight**, especifica el efecto especial (si lo hay) con el que se visualizará el campo. Los valores válidos son los siguientes:

### noHighLight (el valor por omisión)

Indica que no habrá ningún efecto especial; concretamente, no habrá parpadeo, inversión ni subrayado.

### underline

Coloca un subrayado en la parte inferior del campo.

### Conceptos relacionados

- “Enumeraciones en EGL” en la página 484
- “Visión general de las propiedades de EGL” en la página 64

### Consulta relacionada

- “Propiedades de presentación de campos” en la página 66

## inputRequired

La propiedad **inputRequired** indica si es necesario que el usuario especifique datos en el campo. Los valores válidos son *no* (el valor por omisión) y *sí*.

Si el usuario no coloca datos en el campo cuando el valor de propiedad es *sí*, el entorno de ejecución de EGL visualiza un mensaje, como se describe en relación con la propiedad de campo **inputRequiredMsgKey**.

### Conceptos relacionados

- “Formularios de texto” en la página 158

### Consulta relacionada

- “Propiedades de validación” en la página 67
- “validationFailed()” en la página 791
- “Componente DataTable en formato fuente EGL” en la página 473
- “verifyChkDigitMod10()” en la página 913
- “verifyChkDigitMod11()” en la página 914

## inputRequiredMsgKey

La propiedad **inputRequiredMsgKey** identifica el mensaje que se visualiza si la propiedad de campo **inputRequired** se establece como *sí* y el usuario no coloca datos en el campo.

La *tabla de mensajes* (la tabla de datos que contiene el mensaje) está identificada en la propiedad del programa **msgTablePrefix**. Encontrará los detalles sobre el nombre de tabla de datos en el apartado *Componente DataTable en formato fuente EGL*.

El valor de **inputRequiredMsgKey** es una serie o literal que coincide con una entrada de la primera columna de la tabla de mensajes.

Si se utiliza una clave numérica con una tabla de mensajes que espera una clave de caracteres, el número se convierte a una serie de caracteres. Si se utiliza un literal de serie con una tabla de mensajes que espera una clave numérica, el valor de la serie debe ser un entero con firma o sin firma.

#### Conceptos relacionados

“Formularios de texto” en la página 158

#### Consulta relacionada

“Propiedades de validación” en la página 67

“validationFailed()” en la página 791

“Componente DataTable en formato fuente EGL” en la página 473

“verifyChkDigitMod10()” en la página 913

“verifyChkDigitMod11()” en la página 914

## intensity

La propiedad de intensidad, **intensity**, especifica la característica del font visualizado. Los valores válidos son los siguientes:

#### **normalIntensity** (el valor por omisión)

Establece el campo como visible, sin negrita.

#### **bold**

Hace que el texto aparezca en negrita.

#### **dim**

Hace que el texto aparezca con menor intensidad, según sea apropiado cuando un campo de entrada está inhabilitado.

#### **invisible**

Elimina cualquier indicación de que el campo se encuentra en el formulario.

#### Conceptos relacionados

“Enumeraciones en EGL” en la página 484

“Visión general de las propiedades de EGL” en la página 64

#### Consulta relacionada

“Propiedades de presentación de campos” en la página 66

## isBoolean

La propiedad **isBoolean** (antes la propiedad **boolean**) indica que el campo representa un valor booleano. La propiedad restringe los valores de campo válidos y es útil para los formularios de texto e impresión y en PageHandlers, para entrada o salida.

En una página Web asociada con un PageHandler de EGL, un elemento booleano está representado por un recuadro de selección. En un formulario, la situación es la siguiente:

- El valor de un campo numérico es 0 (falso) o 1 (verdadero).

- El valor de un campo de caracteres está representado por una palabra o un subconjunto de una palabra que depende del idioma nacional. En Inglés, por ejemplo, un campo booleano de tres o más caracteres tiene el valor *yes* (verdadero) o *no* (falso) y el valor de un campo booleano de un carácter tiene el valor truncado *y* o *n*.

Los valores específicos de programa para *yes* y *no* quedan determinados por el entorno local.

## isDecimalDigit

La propiedad **isDecimalDigit** determina si debe comprobarse si el valor de entrada incluye solamente dígitos decimales, que son los siguientes:

0123456789

Los valores válidos son *no* (el valor por omisión) y *sí*.

Esta propiedad solamente es aplicable a campos de caracteres.

### Conceptos relacionados

“Formularios de texto” en la página 158

### Consulta relacionada

“Propiedades de validación” en la página 67

“validationFailed()” en la página 791

“Componente DataTable en formato fuente EGL” en la página 473

“verifyChkDigitMod10()” en la página 913

“verifyChkDigitMod11()” en la página 914

## isHexDigit

La propiedad **isHexDigit** determina si debe comprobarse si el valor de entrada incluye solamente dígitos hexadecimales, que son los siguientes:

0123456789abcdefABCDEF

Los valores válidos son *no* (el valor por omisión) y *sí*.

Esta propiedad solamente es aplicable a campos de caracteres.

### Conceptos relacionados

“Formularios de texto” en la página 158

### Consulta relacionada

“Propiedades de validación” en la página 67

“validationFailed()” en la página 791

“Componente DataTable en formato fuente EGL” en la página 473

“verifyChkDigitMod10()” en la página 913

“verifyChkDigitMod11()” en la página 914

## isNullable

La propiedad **isNullable** indica si el elemento puede establecerse en nulo, como es adecuado si la columna de tabla asociada con el elemento puede establecerse en NULL. Los valores válidos son *yes* (el valor por omisión) y *no*.

Para un elemento dado de un registro SQL, están disponibles las siguientes características sólo si **isNullable** se establece en *yes*:

- El programa puede aceptar un valor NULL de la base de datos en el elemento.

- El programa puede utilizar una sentencia **set** para anular el elemento, como se describe en el apartado *set*. El efecto es también inicializar el elemento, tal como se describe en el apartado *Inicialización de datos*.
- El programa puede utilizar una sentencia **if** para probar si el elemento se ha establecido en nulo.

#### Conceptos relacionados

- “Compatibilidad con VisualAge Generator” en la página 439
- “Tipos de registros y propiedades” en la página 135
- “Soporte de SQL” en la página 229
- “Estructura fija” en la página 26
- “Typedef” en la página 27

#### Tareas relacionadas

- “Recuperar datos de tabla SQL” en la página 252

#### Consulta relacionada

- “Propiedades de presentación de campos” en la página 66
- “add” en la página 561
- “close” en la página 568
- “Inicialización de datos” en la página 471
- “delete” en la página 571
- “execute” en la página 574
- “get” en la página 585
- “get next” en la página 597
- “open” en la página 616
- “prepare” en la página 630
- “Tipos primitivos” en la página 34
- “Referencias cruzadas de tipo de registro y tipo de archivo” en la página 737
- “replace” en la página 632
- “set” en la página 636
- “Códigos de datos SQL y variables de lenguaje principal EGL” en la página 744
- “terminalID” en la página 938
- “VAGCompatibility” en la página 405

## isReadOnly

La propiedad **isReadOnly** indica si el elemento y la columna relacionada deben omitirse de las sentencias SQL por omisión que escriben en la base de datos o incluyen una cláusula FOR UPDATE OF. El valor por omisión es *no*; pero EGL trata el elemento de estructura como de “sólo lectura” en estas situaciones:

- La propiedad **key** del registro SQL indica que la columna asociada con el elemento de estructura es una columna de clave; o
- El componente de registro SQL está asociado con más de una tabla; o
- El nombre de columna SQL es una expresión.

#### Conceptos relacionados

- “Compatibilidad con VisualAge Generator” en la página 439
- “Tipos de registros y propiedades” en la página 135
- “Soporte de SQL” en la página 229
- “Estructura fija” en la página 26
- “Typedef” en la página 27

#### Tareas relacionadas

- “Recuperar datos de tabla SQL” en la página 252

### Consulta relacionada

“Propiedades de presentación de campos” en la página 66  
“add” en la página 561  
“close” en la página 568  
“Inicialización de datos” en la página 471  
“delete” en la página 571  
“execute” en la página 574  
“get” en la página 585  
“get next” en la página 597  
“open” en la página 616  
“prepare” en la página 630  
“Tipos primitivos” en la página 34  
“Referencias cruzadas de tipo de registro y tipo de archivo” en la página 737  
“replace” en la página 632  
“set” en la página 636  
“Códigos de datos SQL y variables de lenguaje principal EGL” en la página 744  
“terminalID” en la página 938  
“VAGCompatibility” en la página 405

## lineWrap

La propiedad de EGL **lineWrap** indica si el texto puede acomodarse en una línea nueva siempre que la acomodación es necesaria para no truncar el texto.

Los valores válidos son los de la enumeración **lineWrapType**:

### **character** (el valor por omisión)

El texto de un campo no se dividirá en un espacio en blanco.

### **compress**

El texto de un campo de tipo ConsoleField se dividirá en un espacio en blanco, pero cuando el usuario abandone el campo (navegando a otro campo o pulsando Esc), se eliminarán los espacios extraordinarios utilizados para acomodar texto. Este valor sólo es válido en campos de consola.

### **word**

Si es posible, el texto de un campo se dividirá en un espacio en blanco.

La propiedad **lineWrap** está disponible en los componentes DataItem y es relevante para los campos que aparecen en los siguientes contextos:

- Formularios de consola
- Formularios de impresión
- Formularios de texto
- Páginas Web

La propiedad afecta a la entrada y a la salida.

### Conceptos relacionados

“Enumeraciones en EGL” en la página 484  
“Visión general de las propiedades de EGL” en la página 64

### Consulta relacionada

“Propiedades de formato” en la página 66

## lowerCase

La propiedad de minúsculas, **lowerCase**, indica si deben establecerse caracteres alfabéticos en minúsculas en la entrada de caracteres de un solo byte del usuario. Los valores son los siguientes:

**no (valor predeterminado)**

No establecer la entrada del usuario en minúsculas.

**yes**

Establecer la entrada del usuario en minúsculas.

## masked

La propiedad **masked** indica si un carácter especificado por el usuario se visualizará o no. Esta propiedad se utiliza para especificar contraseñas. Los valores son los siguientes:

**no (valor predeterminado)**

Se visualizará el carácter entrado por el usuario.

**yes**

No se visualizará el carácter entrado por el usuario.

## maxLen

La propiedad **maxLen** especifica la longitud máxima de texto de campo que se escribe en la columna de base de datos. Siempre que sea posible, el valor por omisión para esta propiedad es la longitud del campo; pero el campo es de tipo **STRING**, no existe ningún valor predeterminado.

### Conceptos relacionados

“Compatibilidad con VisualAge Generator” en la página 439

“Tipos de registros y propiedades” en la página 135

“Soporte de SQL” en la página 229

“Estructura fija” en la página 26

“Typedef” en la página 27

### Tareas relacionadas

“Recuperar datos de tabla SQL” en la página 252

### Consulta relacionada

“Propiedades de presentación de campos” en la página 66

“add” en la página 561

“close” en la página 568

“Inicialización de datos” en la página 471

“delete” en la página 571

“execute” en la página 574

“get” en la página 585

“get next” en la página 597

“open” en la página 616

“prepare” en la página 630

“Tipos primitivos” en la página 34

“Referencias cruzadas de tipo de registro y tipo de archivo” en la página 737

“replace” en la página 632

“set” en la página 636

“Códigos de datos SQL y variables de lenguaje principal EGL” en la página 744

“terminalID” en la página 938

“VAGCompatibility” en la página 405

## minimumInput

La propiedad **minimumInput** indica el número mínimo de caracteres que el usuario debe especificar en el campo, si el usuario especifica datos en el campo. El valor por omisión es *0*.

Si el usuario especifica menos caracteres que el número mínimo, el entorno de ejecución de EGL visualiza un mensaje, como se describe en relación con la propiedad de campo **minimumInputMsgKey**.

### Conceptos relacionados

“Formularios de texto” en la página 158

### Consulta relacionada

“Propiedades de validación” en la página 67

“validationFailed()” en la página 791

“Componente DataTable en formato fuente EGL” en la página 473

“verifyChkDigitMod10()” en la página 913

“verifyChkDigitMod11()” en la página 914

## minimumInputMsgKey

La propiedad **minimumInputMsgKey** identifica el mensaje que se visualiza si el usuario actúa como se indica a continuación:

- Especifica datos en el campo; y
- Especifica menos caracteres que el valor especificado en la propiedad **minimumInputRequired**.

La *tabla de mensajes*, la tabla que contiene el mensaje, está identificada en la propiedad del programa **msgTablePrefix**. Encontrará los detalles sobre el nombre de tabla en *Componente DataTable en formato fuente EGL*.

El valor de **minimumInputMsgKey** es una serie o literal que coincide con una entrada de la primera columna de la tabla de mensajes.

Si se utiliza una clave numérica con una tabla de mensajes que espera una clave de caracteres, el número se convierte a una serie de caracteres. Si se utiliza un literal de serie con una tabla de mensajes que espera una clave numérica, el valor de la serie debe ser un entero con firma o sin firma.

### Conceptos relacionados

“Formularios de texto” en la página 158

### Consulta relacionada

“Propiedades de validación” en la página 67

“validationFailed()” en la página 791

“Componente DataTable en formato fuente EGL” en la página 473

“verifyChkDigitMod10()” en la página 913

“verifyChkDigitMod11()” en la página 914

## modified

Indica si el programa considerará el campo como modificado, independientemente de que el usuario haya cambiado el valor. Para obtener detalles, consulte el apartado *Código de datos modificados y propiedad modified*.

El valor por omisión es *no*.

**Conceptos relacionados**

“Código de datos modificados y propiedad modified” en la página 161

“Visión general de las propiedades de EGL” en la página 64

**Consulta relacionada**

“Componente de formulario en formato fuente EGL” en la página 511

**needsSOSI**

La propiedad **needsSOSI** sólo se utiliza para un *campo multibyte* (un campo de tipo MBCHAR) e indica si EGL realiza una comprobación especial cuando el usuario especifica datos de tipo MBCHAR en un dispositivo ASCII. Los valores válidos son *yes* (el valor por omisión) y *no*. La comprobación determina si la entrada puede convertirse adecuadamente al formato SO/SI.

La propiedad resulta útil ya que, durante la conversión, se suprimen los blancos de cola del final de una serie multibyte para permitir la inserción de delimitadores SO/SI en cada subserie de caracteres de doble byte. Para una correcta conversión, el campo de formulario debe tener al menos dos blancos para cada serie de doble byte en el valor multibyte.

Si se establece **needsSOSI** en *no*, el usuario puede rellenar el campo de entrada, en cuyo caso la conversión trunca los datos sin aviso.

Si, no obstante, se establece **needsSOSI** en *sí*, el resultado es el siguiente cuando el usuario especifica datos multibyte:

- El valor se acepta tal cual porque se han proporcionado suficientes blancos; o bien
- El valor se trunca y el usuario recibe un aviso de mensaje.

Establezca **needsSOSI** en *yes* si la entrada ASCII de datos multibyte del usuario puede utilizarse en un sistema z/OS o iSeries.

**Conceptos relacionados**

“Formularios de texto” en la página 158

**Consulta relacionada**

“Propiedades de validación” en la página 67

“validationFailed()” en la página 791

“Componente DataTable en formato fuente EGL” en la página 473

“verifyChkDigitMod10()” en la página 913

“verifyChkDigitMod11()” en la página 914

**newWindow**

La propiedad **newWindow** indica si debe utilizarse una ventana de navegador nueva cuando el entorno de ejecución EGL presenta una página Web en respuesta a la actividad identificada en la propiedad **action**.

Los valores son los de la enumeración **Boolean**:

**No (valor predeterminado)**

La ventana de navegador actual para visualizar la página

**Yes**

Se utiliza una ventana de navegador nueva.

Si no se especifica la propiedad **action**, la ventana de navegador actual se utiliza para visualizar la página.

#### Conceptos relacionados

“Enumeraciones en EGL” en la página 484

“Visión general de las propiedades de EGL” en la página 64

“PageHandler” en la página 194

#### Tareas relacionadas

“Asociar un registro EGL con un JSP Faces” en la página 200

“Enlazar componente mandato JavaServer Faces con PageHandler EGL” en la página 201

“Crear un componente EGL pageHandler” en la página 191

“Crear un campo de EGL y asociarlo con un JSP Faces” en la página 199

“Utilizar la vista Edición rápida para el código de PageHandler” en la página 202

#### Consulta relacionada

“action” en la página 690

“Propiedades del campo PageHandler” en la página 685

“Propiedades del componente PageHandler” en la página 682

“Componente PageHandler en formato fuente EGL” en la página 679

“Soporte de Page Designer para EGL” en la página 192

## numElementsItem

Cuando se establece en un matriz de campo de estructura, la propiedad **numElementsItem** identifica un campo de PageHandler cuyo valor de tiempo de ejecución especifica el número de elementos de matriz a visualizar. La propiedad sólo se utiliza para la salida y sólo es relevante si se establece en un campo de estructura de registro fijo que tiene un valor de apariciones (occurs) mayor que 1.

El valor de **numElementsItem** es un literal de serie que identifica el nombre de un campo de PageHandler. La propiedad no es válida para una matriz dinámica, que incluye un indicador del número de elementos que están utilizándose; para obtener información detallada, consulte el apartado *Matrices*.

#### Conceptos relacionados

“Componentes de registro fijo” en la página 133

“Visión general de las propiedades de EGL” en la página 64

“PageHandler” en la página 194

#### Tareas relacionadas

“Asociar un registro EGL con un JSP Faces” en la página 200

“Crear un componente EGL pageHandler” en la página 191

“Crear un campo de EGL y asociarlo con un JSP Faces” en la página 199

“Utilizar la vista Edición rápida para el código de PageHandler” en la página 202

#### Consulta relacionada

“Matrices” en la página 74

“Propiedades del campo PageHandler” en la página 685

“Propiedades del componente PageHandler” en la página 682

“Componente PageHandler en formato fuente EGL” en la página 679

“Soporte de Page Designer para EGL” en la página 192

## numericSeparator

La propiedad **numericSeparator** indica si debe colocarse un carácter en un número que tenga un componente entero de más de 3 dígitos. Si el separador numérico es una coma, por ejemplo, mil se muestra como *1,000* y un millón se muestra como *1,000,000*. Los valores son los siguientes:

**no (valor predeterminado)**

No utilizar un separador numérico.

**yes**

Utilizar un separador numérico.

El valor por omisión queda determinado por el entorno local de la máquina.

## outline

La propiedad de perfilado, **outline**, le permite trazar líneas en los bordes de campos de cualquier dispositivo que dé soporte a caracteres de doble byte. Los valores válidos son los siguientes:

**box**

Trazar líneas para crear un recuadro alrededor del contenido del campo

**noOutline (el valor por omisión)**

No se trazan líneas

Además, puede especificar cualquiera de los componentes de un recuadro o todos ellos. En este caso, coloque corchetes encerrando uno o varios valores, con cada valor separado del siguiente por una coma, como en este ejemplo:

```
outline = [left, over, right, under]
```

Los valores parciales son los siguientes:

**left**

Trazar una línea vertical en el borde izquierdo del campo

**over**

Trazar una línea horizontal en el borde superior del campo

**right**

Trazar una línea vertical en el borde derecho del campo

**under**

Trazar una línea horizontal en el borde inferior del campo

El contenido de cada campo de formulario va precedido de un byte de atributo. Tenga en cuenta que no puede colocar un byte de atributo en la última columna de un formulario y esperar que aparezca un valor de contorno en la siguiente columna, que está más allá del borde del formulario. (El campo no se acomoda en la línea siguiente.) De forma similar, no puede colocar un byte de atributo en la primera columna de un formulario y esperar que aparezca el valor de contorno en esa columna; el valor de contorno solamente puede aparecer en la siguiente columna.

### Conceptos relacionados

“Enumeraciones en EGL” en la página 484

“Visión general de las propiedades de EGL” en la página 64

### Consulta relacionada

“Propiedades de presentación de campos” en la página 66

## pattern

Empareja el texto introducido por el usuario respecto a un patrón especificado, a efectos de validación.

### Conceptos relacionados

“Visión general de las propiedades de EGL” en la página 64

### Consulta relacionada

“Componente de formulario en formato fuente EGL” en la página 511

## persistent

La propiedad **persistent** indica si el campo se incluye en las sentencias SQL implícitas generadas para el registro SQL. Si el valor es *yes*, se produce un error en tiempo de ejecución en este caso:

- El código se basa en una sentencia SQL implícita y
- Ninguna columna coincide con el valor de la propiedad **column** específica de campo. (El valor por omisión es el nombre de campo.)

Establezca **persistent** en *no* si desea asociar una variable de programa temporal con una fila SQL sin tener una columna correspondiente para la variable en la base de datos. Una variable puede ser deseable, por ejemplo, para indicar si el programa ha modificado la fila.

Para obtener detalles acerca de las sentencias SQL implícitas, consulte la sección *Soporte SQL*.

### Conceptos relacionados

“Compatibilidad con VisualAge Generator” en la página 439

“Tipos de registros y propiedades” en la página 135

“Soporte de SQL” en la página 229

“Estructura fija” en la página 26

“Typedef” en la página 27

### Tareas relacionadas

“Recuperar datos de tabla SQL” en la página 252

### Consulta relacionada

“Propiedades de presentación de campos” en la página 66

“add” en la página 561

“close” en la página 568

“Inicialización de datos” en la página 471

“delete” en la página 571

“execute” en la página 574

“get” en la página 585

“get next” en la página 597

“open” en la página 616

“prepare” en la página 630

“Tipos primitivos” en la página 34

“Referencias cruzadas de tipo de registro y tipo de archivo” en la página 737

“replace” en la página 632

“set” en la página 636

“Códigos de datos SQL y variables de lenguaje principal EGL” en la página 744

“terminalID” en la página 938

“VAGCompatibility” en la página 405

## protect

Especifica si el usuario puede acceder al campo. Los valores válidos son los siguientes:

### **no (el valor por omisión para los campos de variable)**

Establece el campo de forma que el usuario pueda sobrescribir el valor en él.

### **skip (el valor por omisión para los campos de constante)**

Establece el campo de forma que el usuario no pueda sobrescribir el valor en él. Además, el cursor pasa por alto el campo en cualquiera de estos casos:

- El usuario trabaja en el campo anterior por orden de tabulación y pulsa el **tabulador** o rellena ese campo anterior con contenido; o
- El usuario trabaja en el campo siguiente por orden de tabulación y pulsa **Mayúsculas Tabulador**.

### **yes**

Establece el campo de forma que el usuario no pueda escribir encima del valor que haya en él.

### **Conceptos relacionados**

“Visión general de las propiedades de EGL” en la página 64

### **Consulta relacionada**

“Componente de formulario en formato fuente EGL” en la página 511

## selectFromListItem

La propiedad **selectFromListItem** identifica la matriz o la columna de DataTable desde la que el usuario selecciona un valor o varios valores, los cuales se transfieren a la matriz o campo primitivo que se está declarando. El valor que se asigna a **selectFromListItem** se utiliza como valor predeterminado cuando se coloca la matriz o el campo primitivo en la página Web de Page Designer.

El valor de la propiedad **selectFromListItem** es un literal de serie que identifica la matriz origen o la columna de DataTable.

Si especifica esta propiedad al declarar una matriz, el usuario puede seleccionar múltiples valores. Si se especifica esta propiedad al declarar un campo primitivo, el usuario sólo puede seleccionar un valor.

Cualquier valor recibido del usuario debe corresponder a uno de estos tipos:

- El contenido del elemento de matriz o columna de DataTable que ha seleccionado el usuario; o bien
- Una matriz o índice de DataTable, que es un entero que identifica qué elemento o columna se ha seleccionado. El índice puede estar entre 1 y el número de elementos disponibles.

La propiedad **selectType** indica el tipo de valor a recibir, ya haya seleccionado el contenido el usuario o un índice de una matriz o columna.

### **Conceptos relacionados**

“DataTable” en la página 146

“Visión general de las propiedades de EGL” en la página 64

“PageHandler” en la página 194

### Tareas relacionadas

- “Asociar un registro EGL con un JSP Faces” en la página 200
- “Crear un componente EGL pageHandler” en la página 191
- “Crear un campo de EGL y asociarlo con un JSP Faces” en la página 199
- “Utilizar la vista Edición rápida para el código de PageHandler” en la página 202

### Consulta relacionada

- “Matrices” en la página 74
- “Propiedades del campo PageHandler” en la página 685
- “Propiedades del componente PageHandler” en la página 682
- “Componente PageHandler en formato fuente EGL” en la página 679
- “Soporte de Page Designer para EGL” en la página 192
- “selectType”

## selectType

La propiedad **selectType** indica la clase de valor que se recupera en la matriz o campo primitivo que se declara. El valor que se asigna se utiliza como valor predeterminado cuando se coloca la matriz o el campo primitivo en la página Web de Page Designer.

El valor es de la enumeración **selectTypeKind**:

### index (el valor predeterminado)

La matriz o campo primitivo que se declara recibirá índices como respuesta a una selección de usuario. En este caso, la matriz o campo primitivo debe ser de tipo numérico.

### value

La matriz o campo primitivo que se declara recibirá el valor de selección del usuario. En este caso, el elemento puede ser de cualquier tipo.

Para obtener información preparatoria, consulte la propiedad **selectFromListItem**.

### Conceptos relacionados

- “DataTable” en la página 146
- “Visión general de las propiedades de EGL” en la página 64
- “PageHandler” en la página 194

### Tareas relacionadas

- “Asociar un registro EGL con un JSP Faces” en la página 200
- “Crear un componente EGL pageHandler” en la página 191
- “Crear un campo de EGL y asociarlo con un JSP Faces” en la página 199
- “Utilizar la vista Edición rápida para el código de PageHandler” en la página 202

### Consulta relacionada

- “Matrices” en la página 74
- “Propiedades del campo PageHandler” en la página 685
- “Propiedades del componente PageHandler” en la página 682
- “Componente PageHandler en formato fuente EGL” en la página 679
- “Soporte de Page Designer para EGL” en la página 192
- “selectFromListItem” en la página 711

## sign

La propiedad de signo, **sign**, indica la posición en la que se visualiza un signo positivo (+) o negativo (-) cuando se coloca un número en el campo, ya sea desde entrada de usuario o desde el programa. Los valores son los siguientes:

**none**

No se visualiza un signo.

**leading**

El valor por omisión: se visualiza un signo a la izquierda del primer dígito en el número, con la posición exacta del signo determinada por la propiedad **zeroFormat** (descrita más adelante).

**trailing**

Se visualiza un signo justo a la derecha del último dígito del número.

## sqlDataCode

El valor de la propiedad **sqlDataCode** es un número que identifica el tipo de datos SQL que están asociados con el elemento de registro. El sistema de gestión de bases de datos utiliza el código de datos cuando el usuario accede a dichos sistemas durante la declaración, la validación o la ejecución del programa generado.

La propiedad **sqlDataCode** sólo está disponible si ha configurado el entorno para la compatibilidad con VisualAge Generator. Para obtener detalles, consulte el apartado *Compatibilidad con VisualAge Generator*.

El valor por omisión depende del tipo primitivo y de la longitud del elemento de registro, como se muestra en la tabla siguiente. Para obtener más detalles, consulte el apartado *Códigos de datos SQL*.

Tipo primitivo EGL	Longitud	Código de datos SQL
BIN	4	501
	9	497
CHAR	<=254	453
	>254 y <=4000	449
	>4000	457
DBCHAR	<=127	469
	>127 y <=2000	465
	>2000	473
DECIMAL	cualquiera	485
HEX	cualquiera	481
UNICODE	<=127	469
	>127 y <=2000	465
	>2000	473

### Conceptos relacionados

“Compatibilidad con VisualAge Generator” en la página 439

“Tipos de registros y propiedades” en la página 135

“Soporte de SQL” en la página 229

“Estructura fija” en la página 26

“Typedef” en la página 27

### Tareas relacionadas

“Recuperar datos de tabla SQL” en la página 252

### Consulta relacionada

“Propiedades de presentación de campos” en la página 66  
“add” en la página 561  
“close” en la página 568  
“Inicialización de datos” en la página 471  
“delete” en la página 571  
“execute” en la página 574  
“get” en la página 585  
“get next” en la página 597  
“open” en la página 616  
“prepare” en la página 630  
“Tipos primitivos” en la página 34  
“Referencias cruzadas de tipo de registro y tipo de archivo” en la página 737  
“replace” en la página 632  
“set” en la página 636  
“Códigos de datos SQL y variables de lenguaje principal EGL” en la página 744  
“terminalID” en la página 938  
“VAGCompatibility” en la página 405

## sqlVariableLen

El valor de la propiedad **sqlVariableLen** (antes la propiedad **sqlVar**) indica si los blancos finales y los nulos de un campo de caracteres se truncan antes de que el entorno de ejecución EGL escriba los datos en una base de datos SQL. Esta propiedad no tiene ningún efecto sobre los datos que no son de tipo carácter.

Especifique *yes* si la columna de tabla SQL correspondiente es del tipo de datos SQL varchar o vargraphic.

### Conceptos relacionados

“Compatibilidad con VisualAge Generator” en la página 439  
“Tipos de registros y propiedades” en la página 135  
“Soporte de SQL” en la página 229  
“Estructura fija” en la página 26  
“Typedef” en la página 27

### Tareas relacionadas

“Recuperar datos de tabla SQL” en la página 252

### Consulta relacionada

“Propiedades de presentación de campos” en la página 66  
“add” en la página 561  
“close” en la página 568  
“Inicialización de datos” en la página 471  
“delete” en la página 571  
“execute” en la página 574  
“get” en la página 585  
“get next” en la página 597  
“open” en la página 616  
“prepare” en la página 630  
“Tipos primitivos” en la página 34  
“Referencias cruzadas de tipo de registro y tipo de archivo” en la página 737  
“replace” en la página 632  
“set” en la página 636

“Códigos de datos SQL y variables de lenguaje principal EGL” en la página 744  
“terminalID” en la página 938  
“VAGCompatibility” en la página 405

## timeFormat

La propiedad **timeFormat** identifica el formato de las horas.

Los valores válidos son los siguientes:

*patrón*

El valor de *patrón* consiste en un conjunto de caracteres, tal como se describe en la sección *Especificadores de formato de fecha, hora e indicación de la hora*.

Los caracteres pueden eliminarse desde el inicio o el final de una especificación de hora completa, pero no desde el medio.

### defaultTimeFormat

El valor predeterminado en un entorno Java lo establece el entorno local Java.

### eurTimeFormat

El patrón *HH.mm.ss*, que es el formato de hora estándar europeo de IBM.

### isoTimeFormat

El patrón *HH.mm.ss*, que es el formato de hora especificado por International Standards Organization (ISO).

### jisTimeFormat

El patrón *HH:mm:ss*, que es el formato de hora estándar industrial japonés.

### usaTimeFormat

El patrón *hh:mm AM*, que es el formato de hora estándar de EE.UU. de IBM.

La propiedad está disponible en los componentes DataItem y es relevante para los campos que aparecen en los siguientes contextos:

- Formularios de consola
- Formularios de impresión
- Formularios de texto
- Páginas Web

La propiedad se utiliza para la entrada y la salida, pero no en los siguientes casos:

- El campo tiene posiciones decimales, un símbolo de moneda, un separador numérico o un signo; o bien
- El campo es de tipo DBCHAR, MBCHAR o HEX; o bien
- El campo no tiene la longitud suficiente para contener un valor que refleje la máscara. Para conocer otros detalles, consulte *Consideraciones sobre longitud para horas*.)

## Consideraciones sobre longitud para horas

En un formulario, la longitud del campo debe coincidir con la longitud de la máscara de hora que especifique. En un campo de página, las reglas son las siguientes:

- La longitud del elemento debe ser suficiente para la máscara de hora que especifique, pero puede ser más larga
- En el caso de un elemento numérico, los caracteres separadores se excluyen del cálculo de la longitud.

## Consideraciones de E/S para horas

Los datos entrados en un campo de variable se comprueban para asegurarse de que la hora se ha entrado en el formato especificado. El usuario no tiene que entrar los ceros iniciales para horas, minutos y segundos, sino que puede especificar (por ejemplo) 8:15 en lugar de 08:15. El usuario que omite los caracteres separadores, no obstante, debe entrar todos los ceros iniciales.

Una hora almacenada en formato interno no se reconoce como hora, sino simplemente como datos. Si se mueve un campo de hora de 6 caracteres a un elemento de caracteres de longitud 10, por ejemplo, EGL rellena el campo de destino con blancos. No obstante, cuando el valor de 6 caracteres se presenta en un formulario, la hora se convierte desde su formato interno, como corresponda.

### Conceptos relacionados

“Propiedades de tiempo de ejecución Java” en la página 347

“Visión general de las propiedades de EGL” en la página 64

### Consulta relacionada

“Especificadores de fecha, hora e indicación de la hora” en la página 45

“Propiedades de formato” en la página 66

“Propiedades de ejecución de Java (detalles)” en la página 540

## timeStampFormat

La propiedad **timeStampFormat** identifica el formato para indicaciones de la hora que se visualizan en un formulario o que se mantienen en un PageHandler.

Los valores válidos son los siguientes:

*patrón*

El valor de *patrón* consiste en un conjunto de caracteres, tal como se describe en la sección *Especificadores de formato de fecha, hora e indicación de la hora*.

Los caracteres pueden eliminarse desde el inicio o el final de una especificación de indicación de la hora completa, pero no desde el medio.

### defaultTimeStampFormat

En un entorno Java el valor predeterminado lo establece el entorno local Java.

### db2TimestampFormat

El patrón *aaaa-MM-dd-HH.mm.ss.ffffff*, que es el formato de indicación de la hora por omisión de IBM DB2.

### odbcTimestampFormat

El patrón *aaaa-MM-dd HH:mm:ss.ffffff*, que es el formato de indicación de la hora de ODBC.

### Conceptos relacionados

“Propiedades de tiempo de ejecución Java” en la página 347

“Visión general de las propiedades de EGL” en la página 64

### Consulta relacionada

“Especificadores de fecha, hora e indicación de la hora” en la página 45

“Propiedades de ejecución de Java (detalles)” en la página 540

## typeChkMsgKey

La propiedad **typeChkMsgKey** identifica el mensaje que se visualiza si los datos de entrada no son adecuados para el tipo de campo.

La *tabla de mensajes*, la tabla que contiene el mensaje, está identificada en la propiedad del programa **msgTablePrefix**. Encontrará los detalles sobre el nombre de tabla en *Componente DataTable en formato fuente EGL*.

El valor de **typeChkMsgKey** es una serie o literal que coincide con una entrada de la primera columna de la tabla de mensajes.

Si se utiliza una clave numérica con una tabla de mensajes que espera una clave de caracteres, el número se convierte a una serie de caracteres. Si se utiliza un literal de serie con una tabla de mensajes que espera una clave numérica, el valor de la serie debe ser un entero con firma o sin firma.

#### **Conceptos relacionados**

“Formularios de texto” en la página 158

#### **Consulta relacionada**

“Propiedades de validación” en la página 67

“validationFailed()” en la página 791

“Componente DataTable en formato fuente EGL” en la página 473

“verifyChkDigitMod10()” en la página 913

“verifyChkDigitMod11()” en la página 914

## **upperCase**

La propiedad de mayúsculas, **upperCase**, indica si deben establecerse caracteres alfabéticos en mayúsculas en la entrada de caracteres de un solo byte del usuario.

Esta propiedad es de utilidad en formularios y en PageHandlers.

Los valores de **upperCase** son los siguientes:

#### **No (valor predeterminado)**

No establecer la entrada del usuario en mayúsculas.

#### **Yes**

Establecer la entrada del usuario en mayúsculas.

## **validationOrder**

La propiedad **validationOrder** indica cuándo se ejecuta la función de validador del campo en relación con la función de validador de cualquier otro campo. La propiedad es importante si la validación de un campo depende de la validación previa de otro.

El valor es un entero literal.

La validación se produce primero para los campos para los que haya especificado un valor para la propiedad **validationOrder** y los elementos con los valores de numeración inferior se validan primero. La validación se produce entonces para los campos para los que no haya especificado un valor para **validationOrder** y, en este caso, el orden de validación es el orden en que se definen los campos en el PageHandler.

#### **Conceptos relacionados**

“Visión general de las propiedades de EGL” en la página 64

“PageHandler” en la página 194

### Tareas relacionadas

- “Crear un componente EGL pageHandler” en la página 191
- “Crear un campo de EGL y asociarlo con un JSP Faces” en la página 199
- “Utilizar la vista Edición rápida para el código de PageHandler” en la página 202

### Consulta relacionada

- “Propiedades del campo PageHandler” en la página 685
- “Propiedades del componente PageHandler” en la página 682
- “Componente PageHandler en formato fuente EGL” en la página 679
- “Soporte de Page Designer para EGL” en la página 192

## validatorDataTable

La propiedad **validatorDataTable** (antes la propiedad **validatorTable**) identifica una *tabla de validación*, que es un componente dataTable que actúa como base de una comparación con la entrada del usuario. El uso de una tabla de validador se produce después de que el entorno de ejecución de EGL realice las comprobaciones de validación elementales, si las hay. Esas comprobaciones elementales se describen en relación con las siguientes propiedades:

- inputRequired
- isDecimalDigit
- isHexDigit
- minimumInput
- needsSOSI
- validValues

Todas las comprobaciones preceden al uso de la propiedad **validatorFunction**, que especifica una función de validación que realice validación entre valores.

Puede especificar una tabla de validadores de cualquiera de los siguientes tipos, como se describe en *Componente DataTable en formato fuente EGL*:

### matchInvalidTable

Indica que la entrada del usuario debe ser distinta a cualquier valor de la primera columna de la tabla de datos.

### matchValidTable

Indica que la entrada del usuario debe coincidir con un valor de la primera columna de la tabla de datos.

### rangeChkTable

Indica que la entrada del usuario debe coincidir con un valor que esté entre los valores de la primera y segunda columna de al menos una fila de la tabla de datos. (El rango es inclusivo; la entrada del usuario también es válida si coincide con un valor de la primera o segunda columna de cualquier fila.)

Si la validación falla, el mensaje visualizado se basa en el valor de la propiedad **validatorDataTableMsgKey**.

### Conceptos relacionados

- “Formularios de texto” en la página 158

### Consulta relacionada

- “Propiedades de validación” en la página 67
- “validationFailed()” en la página 791
- “Componente DataTable en formato fuente EGL” en la página 473

“verifyChkDigitMod10()” en la página 913

“verifyChkDigitMod11()” en la página 914

## validatorDataTableMsgKey

La propiedad **validatorDataTableMsgKey** (antes la propiedad **validatorTableMsgKey**) identifica el mensaje que se visualiza si el usuario proporciona datos que no corresponden a los requisitos de la *tabla de validadores*, que es la tabla especificada en la propiedad **validatorDataTable**.

La *tabla de mensajes*, la tabla que contiene el mensaje, está identificada en la propiedad del programa **msgTablePrefix**. Encontrará los detalles sobre el nombre de tabla de mensajes en *Componente DataTable en formato fuente EGL*.

El valor de **validatorDataTableMsgKey** es una serie o literal que coincide con una entrada de la primera columna de la tabla de mensajes.

Si se utiliza una clave numérica con una tabla de mensajes que espera una clave de caracteres, el número se convierte a una serie de caracteres. Si se utiliza un literal de serie con una tabla de mensajes que espera una clave numérica, el valor de la serie debe ser un entero con firma o sin firma.

### Conceptos relacionados

“Formularios de texto” en la página 158

### Consulta relacionada

“Propiedades de validación” en la página 67

“validationFailed()” en la página 791

“Componente DataTable en formato fuente EGL” en la página 473

“verifyChkDigitMod10()” en la página 913

“verifyChkDigitMod11()” en la página 914

## validatorFunction

La propiedad **validatorFunction** (antes la propiedad **validator**) identifica una función de validador, que es lógica que se ejecuta después de que el entorno de ejecución de EGL realice comprobaciones de validación elementales, si las hay. Esas comprobaciones se describen en relación con las siguientes propiedades:

- inputRequired
- isDecimalDigit
- isHexDigit
- minimumInput
- needsSOSI
- validValues

Las comprobaciones elementales preceden al uso de la tabla de validadores (como se describe en relación con la propiedad **validatorDataTable**) y todas las comprobaciones preceden al uso de la propiedad **validatorFunction**. Este orden de eventos es importante ya que la función de validador puede realizar la comprobación entre campos y dicha comprobación requiere a menudo valores de campo válidos.

El valor de **validatorFunction** es una función de validación escrita por el usuario. Esta función se codifica sin parámetros y de modo que, si detecta un error, solicita la revisualización del formulario invocando la función `ConverseLib.validationFailed`.

Si la validación falla al especificar una de las dos funciones del sistema, el mensaje visualizado se basa en el valor de la propiedad **validatorFunctionMsgKey**. Si, no obstante, la validación falla al especificar una función de validador propia, la función no utiliza **validatorFunctionMsgKey**, sino que visualiza un mensaje invocando `ConverseLib.validationFailed`.

#### Conceptos relacionados

“Formularios de texto” en la página 158

#### Consulta relacionada

“Propiedades de validación” en la página 67

“validationFailed()” en la página 791

“Componente DataTable en formato fuente EGL” en la página 473

“verifyChkDigitMod10()” en la página 913

“verifyChkDigitMod11()” en la página 914

## validatorFunctionMsgKey

La propiedad **validatorFunctionMsgKey** (antes la propiedad **validatorMsgKey**) identifica un mensaje que se visualiza en el caso siguiente:

- La propiedad **validatorFunction** indica el uso de `sysLib.verifyChkDigitMod10` o `sysLib.verifyChkDigitMod11`; y
- La función especificada indica que la entrada del usuario tiene errores.

La *tabla de mensajes*, la tabla que contiene el mensaje, está identificada en la propiedad del programa **msgTablePrefix**. Encontrará los detalles sobre el nombre de tabla en *Componente DataTable en formato fuente EGL*.

El valor de **validatorFunctionMsgKey** es una serie o literal que coincide con una entrada de la primera columna de la tabla de mensajes.

Si se utiliza una clave numérica con una tabla de mensajes que espera una clave de caracteres, el número se convierte a una serie de caracteres. Si se utiliza un literal de serie con una tabla de mensajes que espera una clave numérica, el valor de la serie debe ser un entero con firma o sin firma.

#### Conceptos relacionados

“Formularios de texto” en la página 158

#### Consulta relacionada

“Propiedades de validación” en la página 67

“validationFailed()” en la página 791

“Componente DataTable en formato fuente EGL” en la página 473

“verifyChkDigitMod10()” en la página 913

“verifyChkDigitMod11()” en la página 914

## validValues

La propiedad **validValues** (antes la propiedad **range**) indica un conjunto de valores válidos para la entrada de usuario. La propiedad se utiliza para campos de caracteres numéricos. El formato de la propiedad es el siguiente:

```
validValues = literalMatriz
```

#### *literalMatriz*

Un literal de matriz de elementos singulares y de dos valores, como en los ejemplos siguientes:

```
validValues = [[1,3], 5, 12]
validValues = ["a", ["bbb", "i"]]
```

Cada elemento singular contiene un valor válido. Cada elemento de dos valores contiene un rango:

- Para los números, el valor situado más a la izquierda es el valor más bajo válido, el situado más a la derecha, el más alto. En el ejemplo anterior, los valores 1, 2 y 3 son válidos para un campo de tipo INT.
- Para los campos de caracteres, la entrada de usuario se compara con el rango de valores para el número de caracteres para el que es posible una comparación. Por ejemplo, el rango ["a", "c"] incluye (como válida) cualquier entrada cuyo primer carácter sea "a", "b" o "c". Aunque la serie "cat" sea mayor que "c" en una secuencia de ordenación, "cat" es una entrada válida.

La regla general es la siguiente: si el primer valor del rango se llama *lowValue* y el segundo *highValue*, la entrada del usuario es válida si se cumple *cualquiera* de estas condiciones:

- La entrada de usuario es igual a *lowValue* o *highValue*
- La entrada de usuario es mayor que *lowValue* y menor que *highValue*
- La serie inicial de caracteres de entrada coincide con la serie inicial de caracteres en *lowValue*, mientras es posible una comparación
- La serie inicial de caracteres de entrada coincide con la serie inicial de caracteres en *highValue*, mientras es posible una comparación

A continuación se proporcionan algunos ejemplos:

```
// los valores válidos son 1, 2, 3, 5, 7, 9 y 11
validValues = [[1, 3], 5, 7, 11]

// los valores válidos son las letras "a" y "z"
validValues = ["a", "z"]

// los valores válidos son las series que empiezan por "a"
validValues = ["a", "a"]

// los valores válidos son cualquier serie
// que empiece por una letra minúscula
validValues = ["a", "z"]
```

Si la entrada del usuario no está dentro del rango especificado, el entorno de ejecución de EGL visualiza un mensaje, como se describe en relación con la propiedad de campo **validValuesMsgKey**.

#### **Conceptos relacionados**

“Formularios de texto” en la página 158

#### **Consulta relacionada**

“Propiedades de validación” en la página 67

“validationFailed()” en la página 791

“Componente DataTable en formato fuente EGL” en la página 473

“verifyChkDigitMod10()” en la página 913

“verifyChkDigitMod11()” en la página 914

## validValuesMsgKey

La propiedad **validValuesMsgKey** (antes la propiedad **rangeMsgKey**) identifica el mensaje que se visualiza si se ha establecido la propiedad de campo **validValues** y el usuario especifica datos fuera de rango en el campo.

La *tabla de mensajes*, la tabla que contiene el mensaje, está identificada en la propiedad del programa **msgTablePrefix**. Encontrará los detalles sobre el nombre de tabla en *Componente DataTable en formato fuente EGL*.

El valor de **validValuesMsgKey** es una serie o literal que coincide con una entrada de la primera columna de la tabla de mensajes.

Si se utiliza una clave numérica con una tabla de mensajes que espera una clave de caracteres, el número se convierte a una serie de caracteres. Si se utiliza un literal de serie con una tabla de mensajes que espera una clave numérica, el valor de la serie debe ser un entero con firma o sin firma.

Esta propiedad solamente es aplicable a campos numéricos.

### Conceptos relacionados

“Formularios de texto” en la página 158

### Consulta relacionada

“Componente DataTable en formato fuente EGL” en la página 473

“validationFailed()” en la página 791

“Propiedades de validación” en la página 67

“verifyChkDigitMod10()” en la página 913

“verifyChkDigitMod11()” en la página 914

## value

La propiedad **value** identifica un literal de tipo serie que se visualiza como contenido del campo cuando se visualiza una página Web. Dicho literal se utiliza como valor predeterminado cuando se coloca un campo de EGL en la página Web de Page Designer.

### Conceptos relacionados

“Visión general de las propiedades de EGL” en la página 64

“PageHandler” en la página 194

### Tareas relacionadas

“Asociar un registro EGL con un JSP Faces” en la página 200

“Crear un campo de EGL y asociarlo con un JSP Faces” en la página 199

“Crear un componente EGL pageHandler” en la página 191

“Utilizar la vista Edición rápida para el código de PageHandler” en la página 202

### Consulta relacionada

“Propiedades del campo PageHandler” en la página 685

“Propiedades del componente PageHandler” en la página 682

“Componente PageHandler en formato fuente EGL” en la página 679

“Soporte de Page Designer para EGL” en la página 192

## zeroFormat

La propiedad de formato de ceros, **zeroFormat**, especifica cómo se visualizan los valores cero en los campos numéricos pero no en campos de tipo MONEY. Esta

propiedad resulta afectada por las propiedades **numeric separator**, **currency** y **fillCharacter**. Los valores de **zeroFormat** son los siguientes:

#### Yes

Se visualiza un valor cero como el número cero, que puede expresarse en comas decimales (0.00 es un ejemplo, si se define el elemento con dos posiciones decimales) y con símbolos de moneda y separadores de caracteres (\$000,000.00 es un ejemplo, dependiendo de los valores de las propiedades **currency** y **numericSeparator**). Las siguientes reglas son aplicables cuando el valor de la propiedad **zeroFormat** es *yes*:

- Si el *carácter de relleno* (el valor de la propiedad **fillCharacter**) es 0, se da formato a los datos con el carácter 0
- Si el carácter de relleno es un nulo, los datos se alinean a la izquierda
- Si el carácter de relleno es un blanco, los datos alinean a la derecha
- Si el carácter de relleno es un asterisco (\*), los asteriscos se utilizan como relleno de la izquierda en lugar de blancos

#### No

Se visualiza un valor cero como una serie del carácter de relleno.

#### Consulta relacionada

“Propiedades de ejecución de Java (detalles)” en la página 540

“set” en la página 636

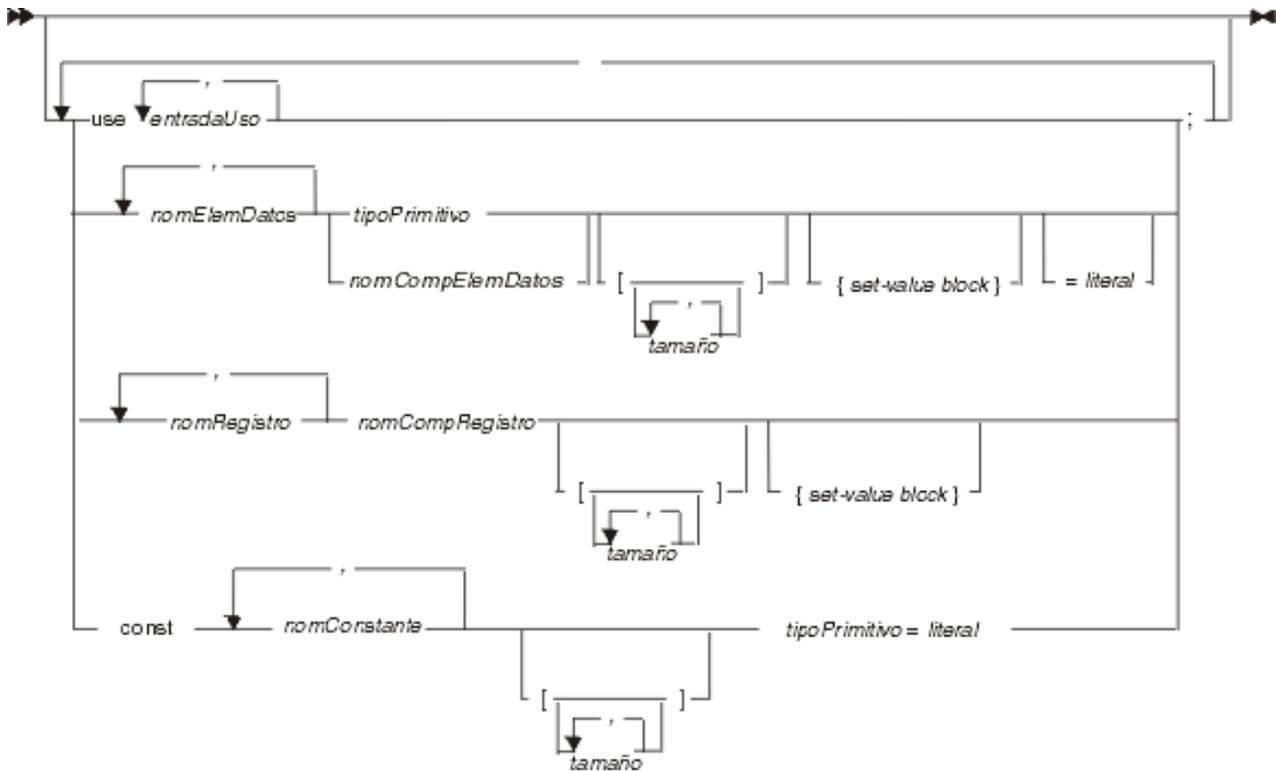
“currencySymbol” en la página 387

“Especificadores de fecha, hora e indicación de la hora” en la página 45

---

## Datos de programa aparte de los parámetros

El diagrama de sintaxis para datos de programa es el siguiente:



**use** *useEntry*

Proporciona un acceso más fácil a una tabla de datos o biblioteca, necesario para acceder a formularios de un grupo de formularios. Para conocer detalles, consulte *Declaración Use*.

*dataItemName*

Nombre de un campo primitivo. Para conocer las reglas de denominación, consulte *Convenios de denominación*.

*primitiveType*

El tipo de un campo primitivo o, en relación con una matriz, el tipo primitivo de un elemento de matriz. En función del tipo, puede ser necesaria la siguiente información:

- La longitud del parámetro o, en relación con una matriz, la longitud de un elemento de matriz. La longitud es un entero que representa el número de caracteres o dígitos del área de memoria.
- Para algunos tipos numéricos puede especificar un entero que represente el número de posiciones después de la coma decimal. La coma decimal no se almacena con los datos.
- Para un elemento de tipo INTERVAL o TIMESTAMP, puede especificar una máscara de fecha y hora, que asigna un significado (como por ejemplo "dígito de año") a una posición dada en el valor de elemento.

Para obtener detalles, consulte el apartado *Tipos primitivos* y el tema correspondiente al tipo determinado.

*dataItemPartName*

El nombre de un componente dataItem que es visible al programa. Para conocer detalles sobre la visibilidad, consulte *Referencias a componentes*.

El componente actúa como un modelo de formato, tal como se describe en *Typedef*.

*size*

El número de elementos de la matriz. Si especifica el número de elementos, la matriz se inicializa con ese número de elementos.

*set-value block*

Para obtener información detallada, consulte los apartados *Visión general de las propiedades de EGL* y *Bloques set-value*.

*recordName*

El nombre de un registro. Para conocer las reglas de denominación, consulte *Convenios de denominación*.

*recordPartName*

El nombre de un componente de registro que es visible al programa. Para conocer detalles sobre la visibilidad, consulte *Referencias a componentes*.

El componente actúa como un modelo de formato, tal como se describe en *Typedef*.

**const** *nombreConstante* *tipoPrimitivo*=*literal*

Nombre, tipo y valor de una constante. Especifique una serie entrecomillada (para un tipo de carácter); un número (para un tipo numérico); o una matriz de valores del tipo adecuado (para una matriz). Ejemplos:

```
const myString String = "Great software!";
const myArray BIN[] = [36, 49, 64];
const myArray02 BIN[][] = [[1,2,3],[5,6,7]];
```

Para conocer las reglas de denominación, consulte *Convenios de denominación*.

### Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13  
“Visión general de las propiedades de EGL” en la página 64  
“Componentes” en la página 17  
“Componente de programa” en la página 139  
“Referencias a variables en EGL” en la página 58  
“Segmentación en aplicaciones de texto” en la página 160  
“Bloques de establecimiento de valor” en la página 67  
“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755  
“Typedef” en la página 27

### Consulta relacionada

“Matrices” en la página 74  
“Inicialización de datos” en la página 471  
“Componente DataItem en formato fuente EGL” en la página 472  
“Componente DataTable en formato fuente EGL” en la página 473  
“Formato fuente EGL” en la página 491  
“Sentencias EGL” en la página 88  
“forward” en la página 583  
“Componente de función en formato fuente EGL” en la página 527  
“Componente de registro indexado en formato fuente EGL” en la página 535  
“Formulario de entrada” en la página 736  
“Registro de entrada” en la página 736  
“INTERVAL” en la página 42  
“Valores de error de E/S” en la página 536  
“Componente de registro MQ en formato fuente EGL” en la página 662  
“Convenios de denominación” en la página 672

“Tipos primitivos” en la página 34

“Componente de registro relativo en formato fuente EGL” en la página 740

“Componente de registro serie en formato fuente EGL” en la página 743

“Componente de registro SQL en formato fuente EGL” en la página 748

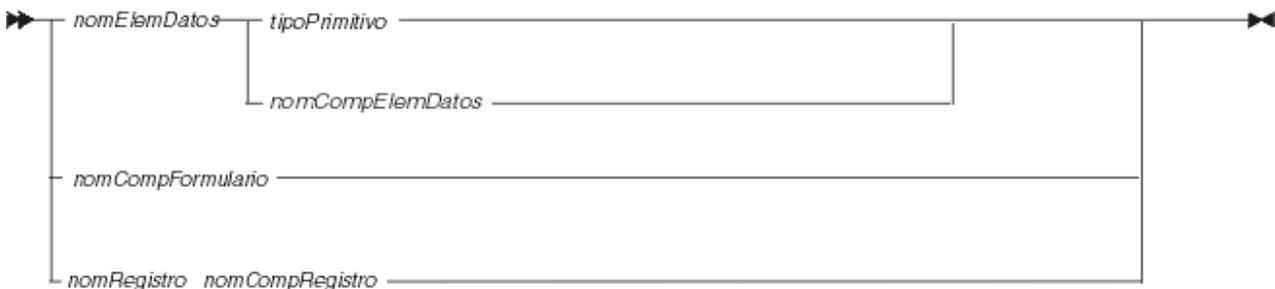
“TIMESTAMP” en la página 44

“Declaración use” en la página 954

---

## Parámetros de programa

El diagrama de sintaxis de un parámetro de programa es el siguiente:



### *nombreElementoDatos*

Nombre de un campo primitivo. Para conocer las normas de denominación, consulte el apartado *Convenios de denominación*.

### *tipoPrimitivo*

Tipo de un campo primitivo. En función del tipo, puede ser necesaria la siguiente información:

- La longitud del parámetro, que es un entero que representa el número de caracteres o dígitos del área de memoria.
- Para algunos tipos numéricos puede especificar un entero que represente el número de posiciones después de la coma decimal. La coma decimal no se almacena con los datos.
- Para un elemento de tipo INTERVAL o TIMESTAMP, puede especificar una máscara de fecha y hora, que asigna un significado (como por ejemplo "dígito de año") a una posición dada en el valor de elemento.

### *nombreComponenteElementoDatos*

El nombre de un componente dataItem que es visible para el programa. Para obtener detalles acerca de la visibilidad, consulte el apartado *Referencias a componentes*.

El componente actúa como modelo de formato, como se describe en *Typedef*.

### *nombreComponenteFormulario*

Nombre de un formulario.

El formulario debe ser accesible a través de un grupo de formularios que esté identificado en una de las declaraciones de uso del programa. Un formulario al que se accede como un parámetro no puede mostrarse al usuario, pero puede proporcionar acceso a valores de campo que se pasan desde otro programa.

Para conocer las normas de denominación, consulte el apartado *Convenios de denominación*.

### *nombreRegistro*

Nombre de un registro o de un registro fijo. Para conocer las normas de denominación, consulte el apartado *Convenios de denominación*.

### *nombreComponenteRegistro*

Nombre de un componente de registro (o de componente de registro fijo) que es visible para el programa. Para obtener detalles acerca de la visibilidad, consulte el apartado *Referencias a componentes*.

El componente actúa como modelo de formato, como se describe en *Typedef*.

Las siguientes consideraciones corresponden a la entrada y salida (E/S) de parámetros de registro:

- Un registro pasado desde otro programa no incluye información de estado del registro, como por ejemplo el valor de error de E/S *endOfFile*. De forma parecida, cualquier cambio en el estado del registro no se devuelve al llamador, de modo que, si realiza una operación de E/S en un parámetro del registro, las pruebas realizadas en ese registro deben producirse antes de que el programa finalice.
- Cualquier operación de E/S realizada en el registro utiliza las propiedades de registro especificadas para el parámetro, no las propiedades de registro especificadas para el argumento.
- Para los registros de tipo *indexedRecord*, *mqRecord*, *relativeRecord* o *serialRecord*, el archivo o cola de mensajes asociada con la declaración de registro se trata como recurso de la unidad de ejecución en lugar de como recurso de programa. Las declaraciones de registro locales comparten el mismo archivo (o cola) siempre que la propiedad de registro **fileName** (o **queueName**) tiene el mismo valor. Sólo puede asociarse un archivo físico simultáneamente con un nombre de archivo o cola, independientemente de cuántos registros estén asociados con el archivo o cola en la unidad de ejecución, y EGL refuerza esta norma cerrando y reabriendo los archivos según convenga.

Un argumento enviado desde otro programa EGL debe tener compatibilidad de referencia con el parámetro relacionado. Para obtener detalles, consulte el apartado *Compatibilidad de referencia en EGL*.

### **Conceptos relacionados**

“Componente de programa” en la página 139

“Referencias a componentes” en la página 21

“Referencias a variables en EGL” en la página 58

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

“Typedef” en la página 27

### **Consulta relacionada**

“Matrices” en la página 74

“Componente de registro básico en formato fuente EGL” en la página 379

“Componente DataItem en formato fuente EGL” en la página 472

“Formato fuente EGL” en la página 491

“Componente de registro indexado en formato fuente EGL” en la página 535

“INTERVAL” en la página 42

“Convenios de denominación” en la página 672

“Tipos primitivos” en la página 34

“Compatibilidad de referencia en EGL” en la página 739

“Componente de registro relativo en formato fuente EGL” en la página 740

“Componente de registro serie en formato fuente EGL” en la página 743

“Componente de registro SQL en formato fuente EGL” en la página 748

“TIMESTAMP” en la página 44

---

## Componente de programa en formato fuente EGL

Un componente de programa se declara en un archivo EGL, como se describe en el apartado *Formato fuente EGL*. Al escribir ese archivo, haga lo siguiente:

- Incluya sólo aquellos componentes que utilice exclusivamente el programa
- No incluya otros componentes principales (dataTable, biblioteca, programa o pageHandler)

El ejemplo siguiente muestra un componente de programa llamado con dos funciones incorporadas, junto con una función autónoma y un componente de registro autónomo:

```
Program myProgram type basicProgram (employeeNum INT)
{
 includeReferencedFunctions = yes
}

// variables globales de programa
employees record_ws;
employeeName char(20);

// función incorporada obligatoria
Function main()
 // inicializar nombres de empleados
 recd_init();

 // obtener el nombre de empleado correcto
 // en función del employeeNum pasado
 employeeName = getEmployeeName(employeeNum);
end

// otra función incorporada
Function recd_init()
 employees.name[1] = "Employee 1";
 employees.name[2] = "Employee 2";
end

// función autónoma
Function getEmployeeName(employeeNum INT) returns (CHAR(20))

 // variable local
 index BIN(4);
 index = 2;
 if (employeeNum > index)
 return("Error");
 else
 return(employees.name[employeeNum]);
 end

end

// componente de registro que actúa como typeDef de empleados
Record record_ws type basicRecord
 10 name CHAR(20)[2];
end
```

para obtener más detalles, consulte el tema correspondiente al tipo determinado de programa.

### Conceptos relacionados

“Componentes” en la página 17

“Componente de programa” en la página 139

### Consulta relacionada

“Programa básico en formato fuente EGL”

“Formato fuente EGL” en la página 491

“Componente de función en formato fuente EGL” en la página 527

“Programa de UI de texto en formato fuente EGL” en la página 731

## Programa básico en formato fuente EGL

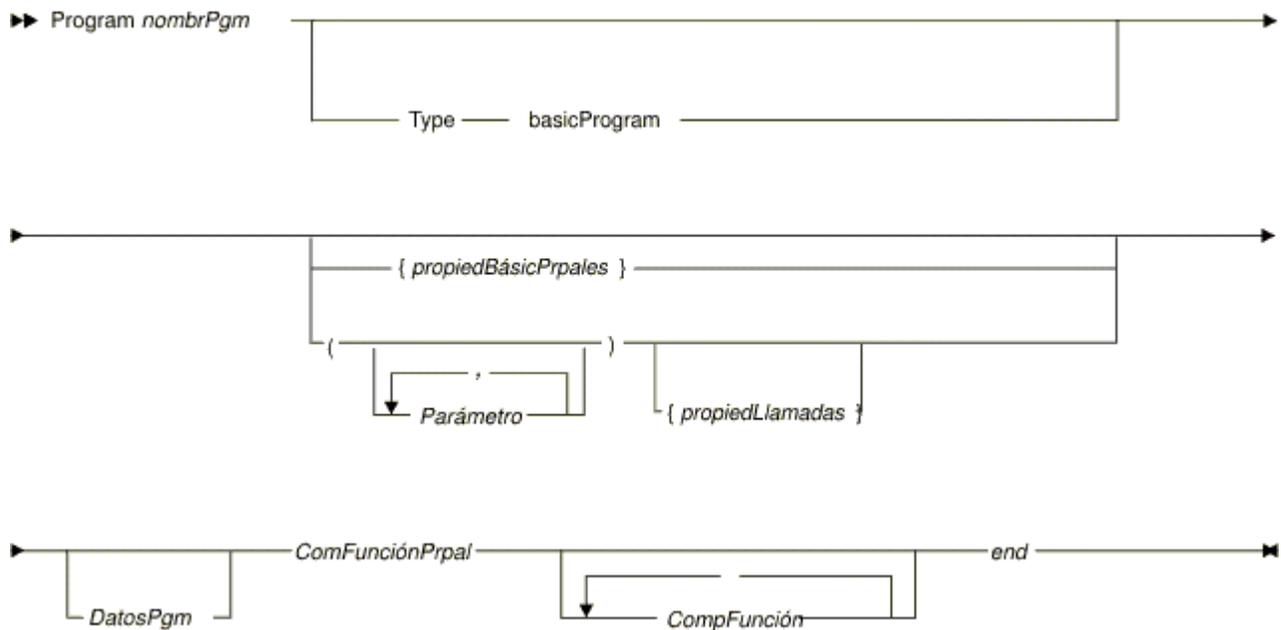
A continuación se ofrece un ejemplo de programa básico:

```
programa myCalledProgram type basicProgram
 (buttonPressed int, returnMessage char(25))

function main()
 returnMessage = "";
 if (buttonPressed == 1)
 returnMessage = "Message1";
 end

 if (buttonPressed == 2)
 returnMessage = "Message2";
 end
end
end
```

El diagrama de sintaxis de un componente de programa de tipo basicProgram es el siguiente:



### Program nombreComponentePrograma ... end

Identifica el componente como componente de programa y especifica el nombre y el tipo. Si el nombre de programa va seguido de un paréntesis de apertura, se trata de un programa básico al que se llama.

Si no establece la propiedad **alias** (como se describe más adelante), el nombre del programa generado es *nombreComponentePrograma*.

Para conocer otras normas, consulte el apartado *Convenios de denominación*.

#### *propiedadesBásicasProgramaPrincipal*

Las propiedades de un programa básico principal son opcionales:

- **alias**
- **allowUnqualifiedItemReferences**
- **handleHardIOErrors**
- **includeReferencedFunctions**
- **inputRecord**
- **localSQLScope**
- **msgTablePrefix**
- **throwNrfEofExceptions**

Para obtener detalles, consulte el apartado *Propiedades de programa*.

#### *parámetro*

Especifica el nombre de un parámetro, que puede ser un elemento de datos, un registro o un formulario; o una matriz dinámica de registros o elementos de datos. Para conocer las normas, consulte el apartado *Convenios de denominación*.

Si el argumento del llamador es una variable (no una constante o literal), los cambios efectuados en el parámetro cambian el área de memoria disponible para el llamador.

Cada parámetro está separado del texto por una coma. Para obtener otros detalles, consulte el apartado *Parámetros de programa*.

#### *propiedadesLlamadas*

Las propiedades llamadas son opcionales:

- **alias**
- **allowUnqualifiedItemReferences**
- **handleHardIOErrors**
- **includeReferencedFunctions**
- **localSQLScope**
- **msgTablePrefix**
- **throwNrfEofExceptions**

Para obtener detalles, consulte el apartado *Propiedades de programa*.

#### *datosPrograma*

Declaraciones de variable y de uso, descritas en el apartado *Datos de programa aparte de los parámetros*.

#### *componenteFunciónPrincipal*

Una función obligatoria denominada *main*, que no toma parámetros. (El único código de programa que puede tomar parámetros es el propio programa y las funciones que no son *main*).

Para obtener detalles acerca de la escritura de funciones, consulte el apartado *Componente de función en formato fuente EGL*.

#### *componenteFunción*

Una función incorporada, que es privada de este programa. Para obtener detalles acerca de la escritura de funciones, consulte el apartado *Componente de función en formato fuente EGL*.

### Conceptos relacionados

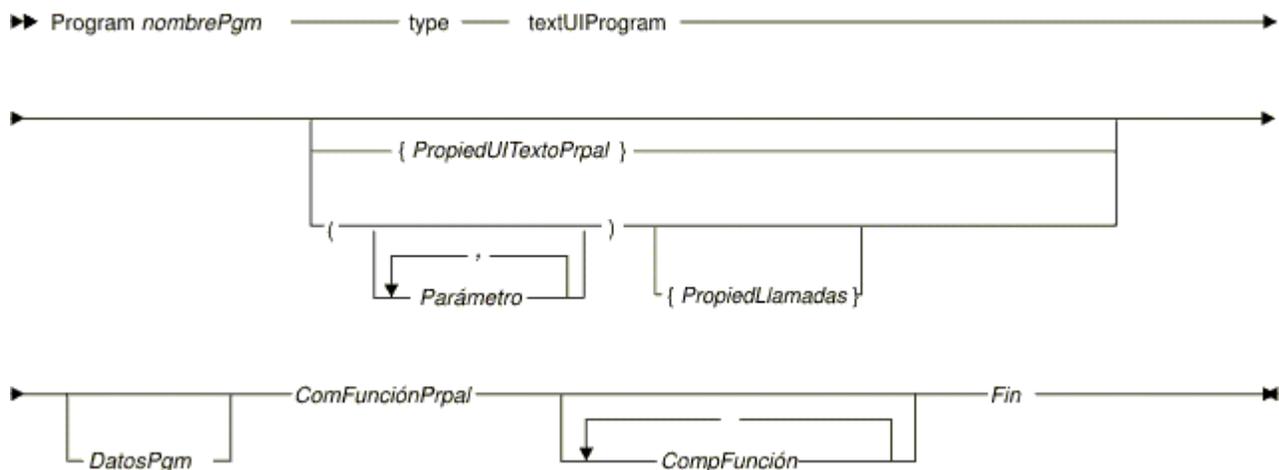
- “Proyectos, paquetes y archivos EGL” en la página 13
- “Visión general de las propiedades de EGL” en la página 64
- “Componentes” en la página 17
- “Componente de programa” en la página 139
- “Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### Consulta relacionada

- “Formato fuente EGL” en la página 491
- “Componente de función en formato fuente EGL” en la página 527
- “Convenios de denominación” en la página 672
- “Datos de programa aparte de los parámetros” en la página 723
- “Parámetros de programa” en la página 726
- “Componente de programa en formato fuente EGL” en la página 728
- “Propiedades de componente de programa” en la página 733
- “Declaración use” en la página 954

## Programa de UI de texto en formato fuente EGL

El diagrama de sintaxis de un componente de programa de tipo textUIProgram es el siguiente:



### Program nombreComponentePrograma ... end

Identifica el componente como componente de programa y especifica el nombre y el tipo. Si el nombre de programa va seguido de un paréntesis de apertura, se trata de un programa básico al que se llama.

Si no establece la propiedad **alias** (como se describe más adelante), el nombre del programa generado es *nombreComponentePrograma*. Si no establece la propiedad **alias** (como se describe más adelante), el nombre del programa generado es *nombreComponentePrograma* o, si genera COBOL, los ocho primeros caracteres de *nombreComponentePrograma*.

Para conocer otras normas, consulte el apartado *Convenios de denominación*.

### propiedadesUITextoPrincipal

Las propiedades de un programa de UI de texto principal son opcionales:

- **alias**
- **allowUnqualifiedItemReferences**
- **handleHardIOErrors**
- **includeReferencedFunctions**

- **inputForm**
- **inputRecord**
- **localSQLScope**
- **msgTablePrefix**
- **segmented**
- **throwNrfEofExceptions**

Para obtener detalles, consulte el apartado *Propiedades de programa*.

#### *parámetro*

Especifica el nombre de un parámetro, que puede ser un elemento de datos, un registro o un formulario; o una matriz dinámica de registros o elementos de datos. Para conocer las normas, consulte el apartado *Convenios de denominación*.

Si el argumento del llamador es una variable (no una constante o literal), los cambios efectuados en el parámetro cambian el área de memoria disponible para el llamador.

Cada parámetro está separado del texto por una coma. Para obtener otros detalles, consulte el apartado *Parámetros de programa*.

#### *propiedadesLlamadas*

Las propiedades llamadas son opcionales:

- **alias**
- **allowUnqualifiedItemReferences**
- **includeReferencedFunctions**
- **msgTablePrefix**

Para obtener detalles, consulte el apartado *Propiedades de programa*.

#### *datosPrograma*

Declaraciones de variable y de uso, descritas en el apartado *Datos de programa aparte de los parámetros*.

#### *componenteFunciónPrincipal*

Una función obligatoria denominada *main*, que no toma parámetros. (El único código de programa que puede tomar parámetros es el propio programa y las funciones que no son *main*).

Para obtener detalles acerca de la escritura de funciones, consulte el apartado *Componente de función en formato fuente EGL*.

#### *componenteFunción*

Una función incorporada, que no está disponible para ningún componente lógico que no sea el programa. Para obtener detalles acerca de la escritura de funciones, consulte el apartado *Componente de función en formato fuente EGL*.

A continuación se ofrece un ejemplo de programa de UI de texto:

```
Program HelloWorld type textUIprogram
{
 use myFormgroup;
 myMessage char(25);

 function main()
 while (ConverseVar.eventKey not pf3)
 myTextForm.msgField = "
 myTextForm.msgField="myMessage";
 converse myTextForm;
 if (ConverseVar.eventKey is pf3)
 exit program;
 end
```

```

 if (ConverseVar.eventKey is pf1)
 myMessage = "Hello Word";
 end
 end
end
end
end

```

### Conceptos relacionados

“Proyectos, paquetes y archivos EGL” en la página 13  
 “Visión general de las propiedades de EGL” en la página 64  
 “Componentes” en la página 17  
 “Componente de programa” en la página 139  
 “Segmentación en aplicaciones de texto” en la página 160  
 “Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### Consulta relacionada

“Formato fuente EGL” en la página 491  
 “Componente de función en formato fuente EGL” en la página 527  
 “Convenios de denominación” en la página 672  
 “Datos de programa aparte de los parámetros” en la página 723  
 “Parámetros de programa” en la página 726  
 “Componente de programa en formato fuente EGL” en la página 728  
 “Propiedades de componente de programa”  
 “Declaración use” en la página 954

---

## Propiedades de componente de programa

Las propiedades de los componentes de programa varían en función de si el programa es llamado o principal y, si es principal, de si el programa es de tipo básico o de UI de texto. Las propiedades son las siguientes:

### **alias** = *"alias"*

Una serie que se incorpora a los nombres de la salida generada. Si no establece la propiedad **alias**, en su lugar se utiliza el nombre del componente programa .

La propiedad **alias** está disponible en cualquier programa.

### **allowUnqualifiedItemReferences** = **no**, **allowUnqualifiedItemReferences** = **yes**

Especifica si debe permitirse al código omitir calificadores de contenedor y subestructura al hacer referencia a elementos de estructuras.

La propiedad **allowUnqualifiedItemReferences** está disponible en cualquier programa.

Observe el siguiente componente de registro, por ejemplo:

```

Record aRecordPart type basicRecord
 10 myItem01 CHAR(5);
 10 myItem02 CHAR(5);
end

```

La variable siguiente se basa en ese componente:

```
myRecord aRecordPart;
```

Si acepta el valor por omisión de **allowUnqualifiedItemReferences** (*no*), debe especificar el nombre de registro al hacer referencia a myItem01, como en esta asignación:

```
myValue = myRecord.myItem01;
```

Si establece la propiedad **allowUnqualifiedItemReferences** en *yes*, sin embargo, puede evitar especificar el nombre de registro:

```
myValue = myItem01;
```

Es aconsejable aceptar el valor por omisión, que promueve una mejor praxis. Al especificar el nombre del contenedor, se reduce la ambigüedad de cara a los usuarios que leen el código y de cara a EGL.

EGL utiliza un conjunto de normas para determinar el área de memoria a la que un nombre de variable o de elemento hace referencia. Para obtener detalles, consulte el apartado *Referencias a variables y constantes*.

#### **handleHardIOErrors = yes, handleHardIOErrors = no**

Establece el valor predeterminado para la variable del sistema **VGVar.handleHardIOErrors**. La variable controla si un programa continúa ejecutándose después de que se haya producido un error grave en una operación de E/S en un bloque try. El valor predeterminado de la propiedad es *yes*, que establece la variable en 1.

El código migrado desde VisualAge Generator no funcionará como antes a menos que establezca **handleHardIOErrors** en *no*, lo que establece la variable en 0.

Esta propiedad está disponible en cualquier programa. Para obtener más detalles, consulte *VGVar.handleHardIOErrors* y *Manejo de excepciones*.

#### **includeReferencedFunctions = no, includeReferencedFunctions = yes**

Indica si el programa contiene una copia de cada función que o se encuentra dentro del programa ni en una biblioteca a la que éste accede.

La propiedad **includeReferencedFunctions** está disponible en cualquier programa.

El valor por omisión es *no*, lo que significa que puede pasar por alto esta propiedad si realiza las siguientes prácticas durante el desarrollo, como se aconseja:

- Coloque las funciones compartidas en una biblioteca
- Coloque las funciones no compartidas en el programa

Si utiliza funciones compartidas que no están en una biblioteca, la generación sólo es posible si establece la propiedad **includeReferencedFunctions** en *yes*.

#### **inputForm = "nombreFormulario"**

Identifica un formulario que se presenta al usuario antes de que se ejecute la lógica del programa, como se describe en el apartado *Formulario de entrada*.

La propiedad **inputForm** sólo está disponible en programas principales de UI de texto.

#### **inputRecord = "registroEntrada"**

Identifica un registro básico global que un programa inicializa automáticamente y que puede recibir datos de un programa que utiliza una sentencia **transfer** para transferir el control. Para obtener más detalles, consulte el apartado *Registro de entrada*.

La propiedad **inputRecord** está disponible en un programa principal.

#### **localSQLScope = yes, localSQLScope = no**

Indica si los identificadores para los conjuntos de resultados SQL y las sentencias preparadas son locales para el programa, lo que constituye el valor

predeterminado. Si acepta el valor *yes*, los programas distintos pueden utilizar independientemente los mismos identificadores.

Si especifica *no*, los identificadores se comparten en toda la unidad de ejecución. Los identificadores creados en el código actual están disponibles en cualquier parte, aunque otro código puede utilizar **localSQLScope = yes** para bloquear el acceso a esos identificadores. Además, el código actual puede hacer referencia a identificadores creados en cualquier parte, pero solo si el otro código ya se ha ejecutado y no ha bloqueado el acceso.

Los efectos de compartir identificadores SQL son los siguientes:

- Puede abrir un conjunto de resultados en un programa y obtener filas de ese conjunto en otro
- Puede preparar una sentencia SQL en un programa y ejecutar esa sentencia en otra

La propiedad **localSQLScope** está disponible en cualquier programa.

#### **msgTablePrefix = "prefijo"**

Especifica el primero de los cuatro caracteres del nombre de la tabla de datos utilizada como tabla de mensajes para el programa. Los demás caracteres del nombre corresponden a uno de los códigos de idioma nacional listados en el apartado *Componente DataTable en formato fuente EGL*.

La propiedad **msgTablePrefix** está disponible en cualquier programa básico o de UI de texto.

Los programas que se ejecutan en aplicaciones Web no utilizan ninguna tabla de mensajes, pero utilizan un recurso de mensajes JavaServer Faces. Para obtener detalles acerca de ese recurso, consulte la descripción de la propiedad **msgResource** en los siguientes apartados:

- *Componente PageHandler en formato fuente EGL*

#### **segmented = no, segmented = yes**

Indica si el programa está segmentado, como se describe en el apartado *Segmentación*. El valor por omisión es *no* en los programas principales de UI de texto. La propiedad no es válida en otros tipos de programas.

#### **throwNrfEofExceptions = no, throwNrfEofExceptions = yes**

Especifica si un error leve provoca el lanzamiento de una excepción. El valor predeterminado es *no*. Para obtener información, consulte la sección *Manejo de excepciones*.

#### **Conceptos relacionados**

“Componente de programa” en la página 139

“Referencias a variables en EGL” en la página 58

“Segmentación en aplicaciones de texto” en la página 160

#### **Consulta relacionada**

“Componente DataTable en formato fuente EGL” en la página 473

“Manejo de excepciones” en la página 94

“forward” en la página 583

“Formulario de entrada” en la página 736

“Registro de entrada” en la página 736

“Convenios de denominación” en la página 672

“Componente PageHandler en formato fuente EGL” en la página 679

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

“handleHardIOErrors” en la página 946

## Formulario de entrada

Cuando se declara un programa principal que se ejecuta en una aplicación de texto, tiene la opción de especificar un *formulario de entrada*, que es un formulario que se presenta al usuario antes de que se ejecute la lógica del programa.

Hay dos escenarios posibles:

- Si el programa es el destino de una sentencia `show-form-returning-to` de un programa generado por EGL, el programa remitente presenta un formulario al usuario y ese formulario debe ser idéntico al formulario de entrada del programa receptor. El programa receptor sólo se invoca una vez que el usuario ha sometido el formulario. Una vez que el usuario ha sometido el formulario, el programa receptor no presenta el formulario de entrada por segunda vez; en lugar de ello, se ejecuta la lógica inicial (la función `execute`).
- Si el programa es el destino de una sentencia `transfer` de un programa (EGL o no EGL) o si el usuario o un mandato del sistema operativo invocan el programa, el programa receptor invierte el formulario de entrada. (En este caso, los campos de entrada del formulario se inicializan antes de visualizarlos). Una vez que el usuario ha sometido el formulario, se ejecuta la lógica inicial (la función `execute`).

El formulario de entrada debe encontrarse en el grupo de formularios especificado en la declaración del componente programa.

### Consulta relacionada

“Inicialización de datos” en la página 471

## Registro de entrada

Cualquier componente de programa puede tener un registro de entrada, que es un registro global que el programa generado por EGL inicializa automáticamente. El registro debe ser de tipo `basicRecord`.

Si el programa se inicia como resultado de una transferencia con registro, inicializa el registro de entrada (que es interno con respecto a ese programa) y, a continuación, asigna los datos transferidos al registro.

Si el registro de entrada es más largo que los datos recibidos, el área adicional del registro de entrada conserva los valores asignados durante la inicialización del registro. Si el registro de entrada es más corto que los datos recibidos, los datos adicionales se truncan.

Si los tipos primitivos de los datos transferidos son incompatibles con los tipos primitivos de las posiciones equivalentes del registro de entrada, el programa receptor puede finalizar de forma anómala.

### Conceptos relacionados

“Visión general de las propiedades de EGL” en la página 64

“Componentes” en la página 17

“Compatibilidad con VisualAge Generator” en la página 439

### Consulta relacionada

“Inicialización de datos” en la página 471

---

## Referencias cruzadas de tipo de registro y tipo de archivo

La tabla siguiente muestra la asociación del tipo de registro y el tipo de archivo según la plataforma destino.

### Conceptos relacionados

“Tipos de registros y propiedades” en la página 135

“Asociaciones de recursos y tipos de archivo” en la página 304

### Tarea relacionada

“Añadir componente asociaciones recursos a archivo de construcción EGL” en la página 309

“Editar componente de asociaciones recursos en archivo construcción EGL” en la página 309

“Eliminar componente asociaciones recursos de archivo construcción EGL” en la página 311

### Consulta relacionada

“resourceAssociations” en la página 398

---

## Propiedades que dan soporte a registros de longitud variable

Al declarar un componente de registro, puede incluir propiedades que den soporte a la utilización de registros de longitud variable. Puede utilizar registros serie de longitud variable para acceder a archivos secuenciales, registros serie o indexados de longitud variable para acceder a archivos VSAM y registros MQ de longitud variable para acceder a colas de mensajes de MQSeries.

### Registros de longitud variable con la propiedad `lengthItem`

La propiedad `lengthItem`, si está presente, identifica un elemento que se utiliza cuando:

- El código lee un registro de un archivo o cola. El elemento de longitud recibe el número de bytes leídos en el registro de longitud variable.
- El código graba un registro. El elemento de longitud especifica el número de bytes que deben añadirse al archivo o cola.

El elemento de longitud puede ser cualquiera de los siguientes:

- Un elemento de estructura del mismo registro
- Un elemento de estructura de un registro que es global al programa o que es local a la función que accede al registro (el elemento de longitud puede estar calificado con una variable de registro declarada en el programa o función)
- Un elemento de datos que sea global con respecto al programa o local con respecto a la función que accede al registro

El elemento de longitud tiene estas características:

- Tiene un tipo primitivo BIN, DECIMAL, INT, NUM o SMALLINT
- No tiene posiciones decimales
- Permite 9 dígitos como máximo

A continuación se ofrece un ejemplo de componente de registro de longitud variable con la propiedad `lengthItem`:

```

Record mySerialRecordPart1 type serialRecord
{
 fileName = "myFile",
 lengthItem = "myOtherField"
}
10 myField01 BIN(4); // 2 bytes de longitud
10 myField02 NUM(3); // 3 bytes de longitud
10 myField03 CHAR(20); // 20 bytes de longitud
end

```

Al grabar un registro, el valor del elemento de longitud debe estar dentro de los límites, a menos que el elemento sea un elemento de carácter. Por ejemplo, un registro de tipo `mySerialRecordPart1` puede tener el elemento de longitud, `myOtherField`, establecido en 2, 5, 6, 7, ... , 24 , 25. Un registro con `myOtherField` establecido en 2 sólo contiene un valor para `myField01`; un registro con `myOtherField` establecido en 5 contiene valores para `myField01` y `myField02`; un registro con `myOtherField` establecido de 6 a 24 también contiene parte de `myField03`.

## Registros de longitud variable con la propiedad `numElementsItem`

La propiedad `NumElementsItem`, si está presente, identifica un elemento que se utiliza cuando el código se añade al archivo o cola o los actualiza. El registro de longitud variable debe tener una matriz como último elemento de estructura de nivel superior. El valor del elemento de número de elementos representa el número real de elementos de matriz que se han escrito. El valor puede ir de 0 al máximo, que es el valor de *apariciones* especificado en la declaración del último elemento de estructura de nivel superior del registro.

El número de bytes escritos es igual a la suma de lo siguiente:

- El número de bytes del componente de longitud fija del registro.
- El valor del elemento de número de elementos multiplicado por el número de bytes de cada elemento de la matriz final.

El elemento de número de elementos tiene estas características:

- Tiene un tipo primitivo BIN, DECIMAL, INT, NUM o SMALLINT
- No tiene posiciones decimales
- Permite 9 dígitos como máximo

A continuación se ofrece un ejemplo de componente de registro de longitud variable con la propiedad `numElementsItem`:

```

Record mySerialRecordPart2 type serialRecord
{
 fileName = "myFile",
 numElementsItem = "myField02"
}
10 myField01 BIN(4); // 2 bytes de longitud
10 myField02 NUM(3); // 3 bytes de longitud
10 myField03 CHAR(20)[3]; // 60 bytes de longitud
 20 mySubField01 CHAR(10);
 20 mySubField02 CHAR(10);
end

```

Escribir un registro del tipo `mySerialRecordPart2` con el elemento de número de elementos `myField02` establecido en 2 da como resultado un registro de longitud variable, grabándose en el archivo o cola `myField01`, `myField02` y dos apariciones de `myField03`.

El elemento de número de elementos debe ser un elemento en la parte de longitud fija del registro de longitud variable. Utilice una referencia no calificada para nombrar el elemento de número de elementos. Por ejemplo, utilice myField02 en lugar de myRecord.myField02.

El elemento de número de elementos no tiene ningún efecto cuando se lee un registro del archivo.

## Registros de longitud variable con las propiedades lengthItem y numElementsItem

Si se especifican las propiedades lengthItem y numElementsItem para un registro de longitud variable, la longitud del registro se calcula utilizando el elemento de número de elementos. La longitud calculada se mueve al elemento de longitud de registro antes de que se escriba el registro en el archivo.

## Registros de longitud variable pasados en una llamada o transferencia

Si en una llamada se pasan registros de longitud variable, se aplican las siguientes normas:

- Se reserva espacio para la longitud máxima especificada para un registro
- Si el valor de la propiedad **type** del elemento callLink es remoteCall o.ejbCall, el elemento de longitud (si existe) debe estar dentro del registro; para obtener detalles, consulte el apartado *Elemento callLink*.

De forma parecida, si se pasan registros de longitud variable en una transferencia, se reserva espacio para la longitud máxima especificada para el registro.

### Conceptos relacionados

“Soporte de MQSeries” en la página 265

“Tipos de registros y propiedades” en la página 135

### Consulta relacionada

“Elemento callLink” en la página 407

“Propiedades de registros MQ” en la página 665

---

## Compatibilidad de referencia en EGL

Un parámetro o una variable es un área de memoria. En algunos casos, la variable contiene los datos empresariales de interés, como por ejemplo, un determinado nombre o ID de empleado. En otros casos, la variable es una *variable de referencia*; contiene un valor (concretamente una dirección de memoria) que se utiliza para acceder a los datos empresariales durante la ejecución.

Cuando se asigna una variable que no es de referencia a otra variable que no es de referencia, el resultado es dos copias de los mismos datos empresariales. Si la variable origen de una sentencia assignment contiene, por ejemplo, un determinado ID de empleado, la sentencia hace que la variable destino también contenga dicho ID. Sin embargo, cuando se asigna una variable de referencia a otra variable de referencia, el resultado es que tanto el origen como el destino contienen un valor que se utiliza para acceder a la misma área de memoria.

Las reglas de compatibilidad de referencia (que se describen más adelante) se aplican en las siguientes situaciones:

- Cuando se asigna una variable de referencia a otra; o bien
- Cuando EGL transfiera datos entre un argumento y el parámetro relacionado, pero sólo cuando se da uno de los siguientes casos:
  - El parámetro de la función receptora tiene el modificador INOUT.
  - El parámetro está en la función onPageLoad de un PageHandler.
  - El parámetro está en un programa EGL que invoca otro programa EGL.
 En estos casos, el argumento es el origen y el parámetro es el destino.

Las reglas de la compatibilidad de referencia son las siguientes:

- Sólo se puede asignar o pasar una variable de referencia a otra variable de referencia del mismo tipo.
- Cuando el origen (o argumento) hace referencia a un tipo primitivo o a una matriz de DataItems, se aplica lo siguiente:
  - Las características primitivas (si existen) deben ser idénticas. Por ejemplo, un argumento de tipo CHAR(6) no es compatible con un parámetro de tipo CHAR(7).
  - Un argumento que tiene posibilidad de nulos es compatible con un parámetro con o sin posibilidad de nulos. Un argumento que no tiene posibilidad de nulos sólo es compatible con un parámetro que no tiene posibilidad de nulos.
- Aparte de los componentes DataItem, los componentes de diferentes paquetes siempre se consideran como tipos distintos.
- En relación con un campo de estructura o registro fijo, la longitud del argumento debe ser mayor o igual que la longitud del parámetro. Esta regla impide que el código receptor acceda a la memoria que no es válida.

#### Conceptos relacionados

“PageHandler” en la página 194

#### Consulta relacionada

“Parámetros de función” en la página 522

“Componente de función en formato fuente EGL” en la página 527

“Componente PageHandler en formato fuente EGL” en la página 679

“Parámetros de programa” en la página 726

“Componente de programa en formato fuente EGL” en la página 728

---

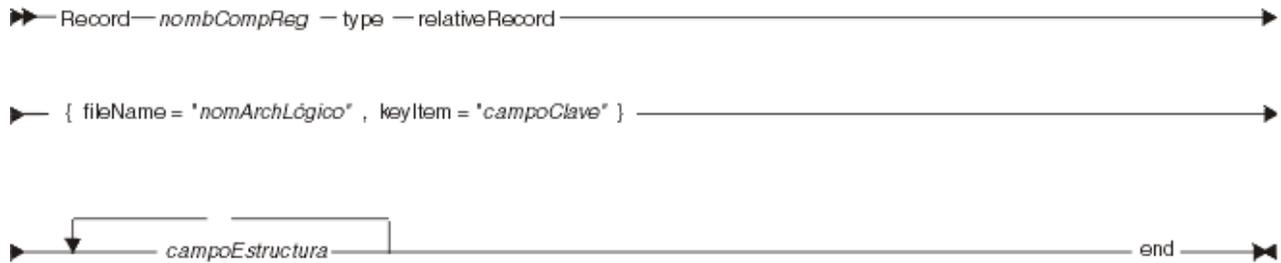
## Componente de registro relativo en formato fuente EGL

Un componente de registro fijo de tipo relativeRecord se declara en un archivo EGL, como se describe en el apartado *Formato fuente EGL*.

A continuación se ofrece un ejemplo de componente de registro relativo:

```
Record myRelativeRecordPart type relativeRecord
{
 fileName = "myFile",
 keyItem = "myKeyItem"
}
10 myKeyItem NUM(4);
10 myContent CHAR(76);
end
```

El diagrama de sintaxis de un componente de registro relativo es el siguiente:



### **Record nombreComponenteRegistro relativeRecord**

Identifica el componente como de tipo `relativeRecord` y especifica el nombre. Para conocer las normas de denominación, consulte el apartado *Convenios de denominación*.

### **fileName = "nombreArchivoLógico"**

El nombre de archivo lógico. Para obtener detalles acerca del significado de la entrada, consulte el apartado *Asociaciones de recursos (visión general)*. Para conocer las normas, consulte el apartado *Convenios de denominación*.

### **keyItem = "campoClave"**

El campo de clave, que puede ser cualquiera de estas áreas de memoria:

- Un campo en el mismo registro fijo
- Una variable o un campo que es global al programa o local a la función que accede al registro fijo

Debe utilizar una referencia no calificada para denominar el campo de clave. Por ejemplo, utilice `myField` en lugar de `myRecord.myField`. (En una función, sin embargo, puede hacer referencia al campo de clave al igual que haría con cualquier campo). El campo de clave debe ser exclusivo en el ámbito local de la función que accede al registro o bien debe estar ausente del ámbito local y ser exclusivo en el ámbito global.

El campo de clave tiene las siguientes características:

- Tiene un tipo primitivo `NUM`, `BIN`, `DECIMAL`, `INT`, `NUM` o `SMALLINT`
- No tiene posiciones decimales
- Permite 9 dígitos como máximo

Sólo las sentencias **get** y **add** utilizan el campo de clave de registro relativo, pero el campo de clave debe estar disponible para cualquier función que utilice el registro para el acceso a archivos.

### *campoEstructura*

Un campo de estructura, como se describe en la sección *Campo de estructura en formato fuente EGL*.

### **Conceptos relacionados**

- “Proyectos, paquetes y archivos EGL” en la página 13
- “Referencias a componentes” en la página 21
- “Componentes” en la página 17
- “Componentes de registro” en la página 132
- “Referencias a variables en EGL” en la página 58
- “Typedef” en la página 27

### **Tareas relacionadas**

- “Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### Consulta relacionada

- “Matrices” en la página 74
- “Componente DataItem en formato fuente EGL” en la página 472
- “Formato fuente EGL” en la página 491
- “Componente de función en formato fuente EGL” en la página 527
- “Componente de registro indexado en formato fuente EGL” en la página 535
- “Componente de registro MQ en formato fuente EGL” en la página 662
- “Convenios de denominación” en la página 672
- “Tipos primitivos” en la página 34
- “Componente de programa en formato fuente EGL” en la página 728
- “Asociaciones de recursos y tipos de archivo” en la página 304
- “Componente de registro serie en formato fuente EGL” en la página 743
- “Componente de registro SQL en formato fuente EGL” en la página 748
- “Elemento de estructura en el formato fuente de EGL” en la página 752

---

## Unidad de ejecución

Una *unidad de ejecución* es un conjunto de programas que están relacionados mediante llamadas locales o (en algunos casos) mediante transferencias. Cada unidad de ejecución tiene las siguientes características:

- Los programas funcionan juntos como un grupo. Cuando se produce un error grave pero no se maneja, todos los programas de la unidad de ejecución se eliminan de la memoria.
- Los programas comparten las mismas propiedades de entorno de ejecución. Por ejemplo, las mismas bases de datos y archivos están disponibles en toda la unidad de ejecución, y cuando invoca `sysLib.connect` o `VGLib.connectionService` para conectarse dinámicamente a una base de datos, la conexión está presente en cualquier programa que recibe el control en la misma unidad de ejecución.

La *unidad de ejecución Java* se compone de programas que se ejecutan en una única hebra. Una nueva unidad de ejecución puede iniciarse con un programa principal, igual que cuando el usuario invoca el programa. Una sentencia **transfer** también invoca un programa principal pero continúa la misma unidad de ejecución.

En los casos siguientes, un programa llamado es el programa inicial de una unidad de ejecución:

- La llamada es una llamada de un bean de sesión EJB; o bien
- La llamada es una llamada remota, excepto que la misma unidad de ejecución continúa en el siguiente caso:
  - EGL o VisualAge Generator genera el programa llamado; y
  - No interviene ninguna escucha TCP/IP en la llamada.

Todos los programas de una unidad de ejecución Java se ven afectados por las mismas propiedades de entorno de ejecución Java.

### Conceptos relacionados

- “Propiedades de tiempo de ejecución Java” en la página 347
- “Componente de opciones de enlace” en la página 311

### Consulta relacionada

- “Base de datos por omisión” en la página 251
- “connect()” en la página 895
- “connectionService()” en la página 916

---

## resultSetID

El identificador de conjunto de resultados está en la sintaxis de EGL y se utiliza cuando se accede a una base de datos relacional y se necesita relacionar los siguientes tipos de sentencias:

- En primer lugar, una sentencia **open** o **get** que selecciona un conjunto de resultados, o bien una sentencia **open** que llama a un procedimiento almacenado que devuelve un conjunto de resultados
- En segundo lugar, las sentencias que acceden al conjunto de resultados

Si utiliza un registro SQL como objeto de E/S, el nombre de registro es suficiente para relacionar un tipo de sentencia con otro, a menos que modifique las sentencias SQL asociadas con el registro para recuperar distintos conjuntos de columnas para actualizar en distintas sentencias. En este caso, utilice un identificador de conjunto de resultados para identificar el conjunto de resultados asociado con una sentencia **replace** de EGL.

### Conceptos relacionados

“Soporte de SQL” en la página 229

### Consulta relacionada

“replace” en la página 632

“open” en la página 616

“get” en la página 585

---

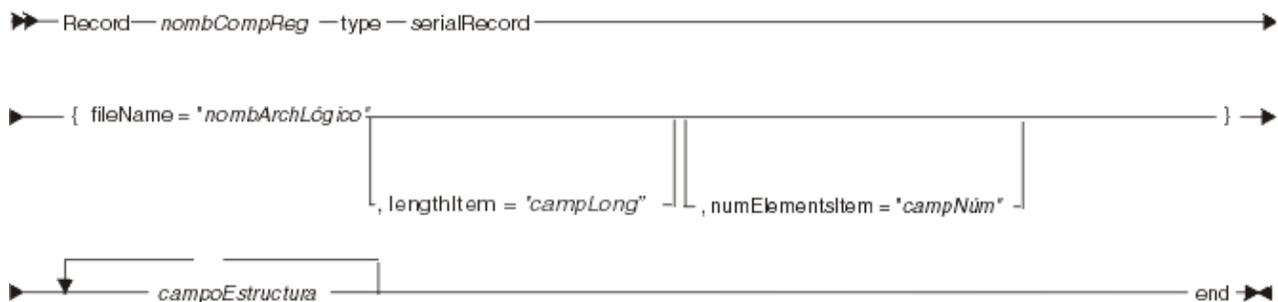
## Componente de registro serie en formato fuente EGL

Un componente de registro de tipo `serialRecord` se declara en un archivo EGL, como se describe en el apartado *Formato fuente EGL*.

A continuación se ofrece un ejemplo de componente de registro serie:

```
Record mySerialRecordPart type serialRecord
{
 fileName = "myFile"
}
10 myField01 CHAR(2);
10 myField02 CHAR(78);
end
```

El diagrama de sintaxis de un componente de registro serie es el siguiente:



### Record nombreComponenteRegistro serialRecord

Identifica el componente como de tipo `serialRecord` y especifica el nombre del componente. Para conocer las normas de denominación, consulte el apartado *Convenios de denominación*.

**fileName** = "*nombreArchivoLógico*"

El nombre de archivo lógico. Para obtener detalles acerca del significado de la entrada, consulte el apartado *Asociaciones de recursos (visión general)*. Para conocer las normas, consulte el apartado *Convenios de denominación*.

**lengthItem** = "*campoLongitud*"

El campo de longitud, como se describe en el apartado *Propiedades que dan soporte a registros de longitud variable*.

**numElementsItem** = "*campoNúm*"

El campo de número de elementos, como se describe en la sección *Propiedades que dan soporte a registros de longitud variable*.

*campoEstructura*

Un campo de estructura, como se describe en la sección *Campo de estructura en formato fuente EGL*.

### **Conceptos relacionados**

"Proyectos, paquetes y archivos EGL" en la página 13

"Referencias a componentes" en la página 21

"Componentes" en la página 17

"Componentes de registro" en la página 132

"Referencias a variables en EGL" en la página 58

"Asociaciones de recursos y tipos de archivo" en la página 304

"Typedef" en la página 27

### **Tareas relacionadas**

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

### **Consulta relacionada**

"Matrices" en la página 74

"Componente DataItem en formato fuente EGL" en la página 472

"Formato fuente EGL" en la página 491

"Componente de función en formato fuente EGL" en la página 527

"Componente de registro indexado en formato fuente EGL" en la página 535

"Componente de registro MQ en formato fuente EGL" en la página 662

"Convenios de denominación" en la página 672

"Tipos primitivos" en la página 34

"Componente de programa en formato fuente EGL" en la página 728

"Propiedades que dan soporte a registros de longitud variable" en la página 737

"Componente de registro relativo en formato fuente EGL" en la página 740

"Componente de registro SQL en formato fuente EGL" en la página 748

"Elemento de estructura en el formato fuente de EGL" en la página 752

---

## **Códigos de datos SQL y variables de lenguaje principal EGL**

La propiedad **SQL data code** identifica el tipo de datos SQL que debe asociarse con la variable de lenguaje principal EGL. El sistema de gestión de bases de datos utiliza el código de datos durante la declaración, la validación o la ejecución del programa generado.

Puede que desee modificar el código de datos SQL para una variable de lenguaje principal de los tipos primitivos CHAR, DBCHAR, HEX o UNICODE. Para una variable de lenguaje principal de uno de los demás tipos primitivos, sin embargo, los códigos de datos SQL son fijos.

Si EGL ha recuperado una definición de columna del sistema de gestión de bases de datos, no modifique el código de datos SQL recuperado, si existe.

Las secciones que siguen describen estos temas:

- “Columnas de longitud fija y variable”
- “Compatibilidad entre los tipos de datos SQL y los tipos primitivos EGL”
- “VARCHAR, VARGRAPHIC y los tipos de datos LONG relacionados” en la página 747
- “DATE, TIME y TIMESTAMP” en la página 747

## Columnas de longitud fija y variable

Para indicar que una columna de tabla es de longitud variable o de longitud fija, establezca el código de datos SQL de la variable de lenguaje principal correspondiente en el valor adecuado, según se muestra en la tabla siguiente.

Tipo primitivo EGL	Tipo de datos SQL	Variable o fija	Código de datos SQL
CHAR	CHAR (valor por omisión)	Fija	453
	VARCHAR, longitud < 255	Variable	449
	VARCHAR, longitud > 254	Variable	457
DBCHAR, UNICODE	GRAPHIC (valor por omisión)	Fija	469
	VARGRAPHIC, longitud < 128	Variable	465
	VARGRAPHIC, longitud > 127	Variable	473

**Nota:** Un tipo de datos SQL puede requerir la utilización de indicadores de nulo, pero este requisito no afecta en absoluto a la forma de codificar un programa EGL. Para obtener detalles acerca de los nulos, consulte el apartado *Soporte SQL*.

## Compatibilidad entre los tipos de datos SQL y los tipos primitivos EGL

Una variable de lenguaje principal EGL y la columna de tabla SQL correspondiente son compatibles en las siguientes situaciones:

- La columna SQL tiene cualquier formato de datos de carácter, y la variable de lenguaje principal EGL es de tipo CHAR con una longitud inferior o igual a la de la columna SQL.
- La columna SQL tiene cualquier formato de datos DBCHAR, y la variable de lenguaje principal EGL es de tipo DBCHAR con una longitud inferior o igual a la de la columna SQL.
- La columna SQL tiene cualquier formato numérico, y la variable de lenguaje principal EGL es de cualquiera de estos tipos:
  - BIN, con 2 o 4 bytes sin posiciones decimales.
  - DECIMAL, con una longitud máxima de 18 dígitos, incluidas posiciones decimales. El número de dígitos de una variable DECIMAL debe ser el mismo para la variable EGL y para la columna.
  - SMALLINT.

- La columna SQL de cualquier tipo de datos, la variable de lenguaje principal EGL es de tipo HEX y la columna y la variable de lenguaje principal contienen el mismo número de bytes. No se realiza ninguna conversión de datos durante la transferencia de datos.

Las variables de lenguaje principal EGL de tipo HEX dan soporte al acceso a cualquier columna SQL cuyo tipo de datos no corresponda a un tipo primitivo EGL.

Si se leen datos de carácter de una columna de tabla SQL en una variable de lenguaje principal más corta, el contenido se trunca por la derecha. Para comprobar el truncamiento, utilice la palabra reservada **trunc** de la sentencia EGL **if**.

Si se leen datos numéricos de una columna de tabla SQL en una variable de lenguaje principal más corta, los ceros iniciales se truncan por la izquierda. Si el número sigue sin caber en la variable de lenguaje principal, se suprimen por la derecha partes fraccionarias del número (en decimales), sin indicación de error. Si el número sigue sin caber, se devuelve un código SQL negativo para indicar una condición de desbordamiento.

La tabla siguiente muestra las características de las variables de lenguaje principal EGL que se asignan cuando la función de recuperación del editor EGL extrae información de un sistema de gestión de bases de datos.

Tipo de datos SQL	Características de la variable de lenguaje principal EGL			Código de datos SQL (SQLTYPE)
	Tipo primitivo	Longitud	Número de bytes	
BIGINT	HEX	16	8	493
CHAR	CHAR	1-32767	1-32767	453
DATE	CHAR	10	10	453
DECIMAL	DECIMAL	1-18	1-10	485
DOUBLE	HEX	16	8	481
FLOAT	HEX	16	8	481
GRAPHIC	DBCHAR	1-16383	2-32766	469
INTEGER	BIN	9	4	497
LONG VARBINARY	HEX	65534	32767	481
LONG VARCHAR	CHAR	>4000	>4000	457
LONG VARGRAPHIC	DBCHAR	>2000	>4000	473
NUMERIC	DECIMAL	1-18	1-10	485
REAL	HEX	8	4	481
SMALLINT	BIN	4	2	501
TIME	CHAR	8	8	453
TIMESTAMP	CHAR	26	26	453
VARBINARY	HEX	2-65534	1-32767	481
VARCHAR	CHAR	≤4000	≤4000	449
VARGRAPHIC	DBCHAR	≤2000	≤4000	465

## **VARCHAR, VARGRAPHIC y los tipos de datos LONG relacionados**

La definición de una columna de tabla SQL de tipo VARCHAR o VARGRAPHIC incluye una longitud máxima, y el mandato de recuperación utiliza ese máximo para asignar una longitud a la variable de lenguaje principal EGL. Sin embargo, la definición de una columna de tabla SQL de tipo LONG VARCHAR o VARGRAPHIC no incluye una longitud máxima, y el mandato de recuperación utiliza el máximo del tipo de datos SQL para asignar una longitud.

## **DATE, TIME y TIMESTAMP**

Asegúrese de que el formato utilizado para el formato Gregoriano largo por omisión del sistema EGL es el mismo que el formato de fecha especificado para el gestor de bases de datos SQL. Para obtener detalles acerca de cómo se establece el formato EGL, consulte el apartado correspondiente a *VGVar.currentFormattedGregorianDate*.

Los dos formatos deben coincidir a fin de que las fechas suministradas por la variable de sistema *VGVar.currentFormattedGregorianDate* estén en el formato esperado por el gestor de bases de datos SQL.

### **Conceptos relacionados**

“Soporte de SQL” en la página 229

### **Consulta relacionada**

“Propiedades de elementos SQL” en la página 67

“currentFormattedGregorianDate” en la página 941

---

## **Diseño interno de los registros SQL**

Debe conocer el diseño interno de un registro SQL en cualquiera de estas situaciones:

- Si utiliza una sentencia EGL assignment para copiar un registro SQL a o desde un registro de un tipo diferente
- El argumento de tiempo de ejecución pasado a un programa EGL es un registro SQL, pero el parámetro de programa no es un registro SQL
- El argumento de tiempo de ejecución pasado a una función EGL es un registro SQL; en este caso, el parámetro debe ser un registro de almacenamiento de trabajo
- Recibe un registro SQL como parámetro en un programa no EGL

Cada elemento de estructura de un registro SQL va precedido de cuatro bytes. Los dos primeros bytes son un indicador de nulo, y un nulo se interpreta como cualquier valor negativo. Los dos segundos bytes se reservan para utilizarlos como campo de longitud y *no* debe acceder a ese campo.

### **Conceptos relacionados**

“Componente de función” en la página 140

“Componente de programa” en la página 139

“Soporte de SQL” en la página 229

### **Consulta relacionada**

“Asignaciones” en la página 374

---

## Componente de registro SQL en formato fuente EGL

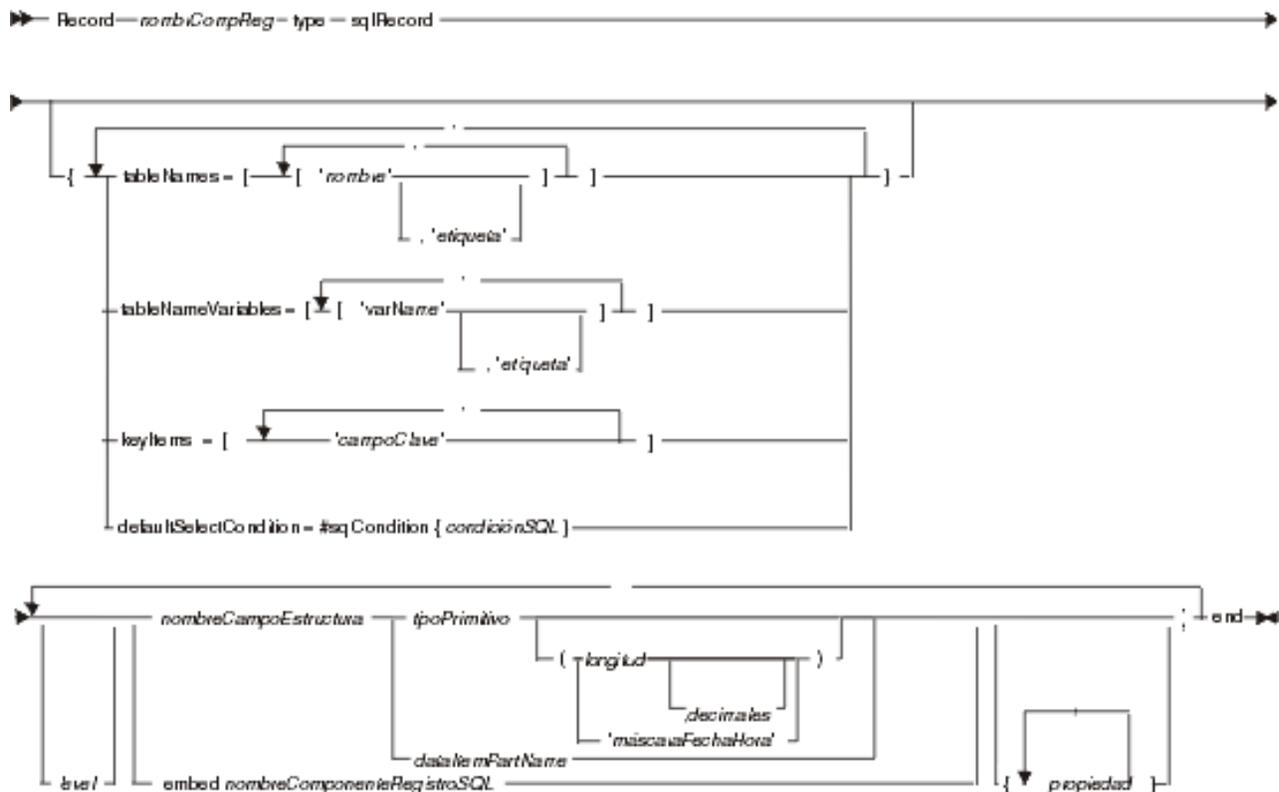
Un componente de registro de tipo `sqlRecord` se declara en un archivo EGL, como se describe en el apartado *Formato fuente EGL*. Para obtener una visión general de la forma en que EGL interactúa con bases de datos relacionales, consulte el apartado *Soporte SQL*.

A continuación se ofrece un ejemplo de componente de registro SQL:

```
Record mySQLRecordPart type sqlRecord
{
 tableNames = [{"mySQLTable", "T1"}],
 keyItems = ["myHostVar01"],
 defaultSelectCondition =
 #sqlCondition{ // sin espacio entre #sqlCondition y la llave
 myHostVar02 = 4 -- iniciar cada comentario SQL
 -- con guión doble
 }
}

// La estructura de un registro SQL no tiene jerarquía
10 myHostVar01 myDataItemPart01
{
 column = "column01",
 isNullable = no,
 isReadOnly = no
};
10 myHostVar02 myDataItemPart02
{
 column = "column02",
 isNullable = yes,
 isReadOnly = no
};
end
```

El diagrama de sintaxis de un componente de registro SQL es el siguiente:



### Record nombreComponenteRegistro sqlRecord

Identifica el componente como componente de registro de tipo `sqlRecord` y especifica el nombre. Para conocer las normas, consulte el apartado *Convenios de denominación*.

#### `tableNames = [{"nombre", "etiqueta"}, ..., [{"nombre", "etiqueta"}]`

Lista la tabla o tablas a las que accede el registro SQL. Si especifica una etiqueta para un nombre de tabla determinado, ésta se incluye en las sentencias SQL por omisión asociadas con el registro.

Puede incluir comillas (") en un nombre de tabla colocando el carácter de escape (\) delante de ellas. Ese convenio es necesario, por ejemplo, cuando un nombre de tabla es una de las siguientes palabras reservadas SQL:

- CALL
- FROM
- GROUP
- HAVING
- INSERT
- ORDER
- SELECT
- SET
- UPDATE
- UNION
- VALUES
- WHERE

Cada uno de estos nombres debe especificarse entre comillas. Si el único nombre de tabla es *SELECT*, por ejemplo, la cláusula *tableNames* será la siguiente:

```
tableNames=[["\SELECT\"]]
```

Una situación similar se produce cuando una de dichas palabras reservadas SQL se utiliza como nombre de columna.

**tableNameVariables = [{"nomVar", "etiqueta"}, ..., {"nomVar", "etiqueta"}]**

Lista una o varias variables de nombre de tabla, cada una de las cuales contiene el nombre de una tabla a la que accede el registro SQL. El nombre de una tabla sólo se determina en tiempo de ejecución.

La variable puede estar calificada por un nombre de biblioteca y puede tener subíndices.

Si especifica una etiqueta para una variable de nombre de tabla determinada, ésta se incluye en las sentencias SQL por omisión asociadas con el registro.

Puede utilizar variables de nombre de tabla aisladas o junto con nombres de tabla, pero la utilización de variables de nombre de tabla garantiza que las características de la sentencia SQL sólo se determinarán en tiempo de ejecución.

Puede incluir comillas (") en una variable de nombre de tabla colocando el carácter de escape (\) delante de ellas.

**keyItems = [{"elemento"}, ..., {"elemento"}]**

Indica que la columna asociada con un elemento de registro determinado forma parte de la clave de la tabla de base de datos. Si la tabla de base de datos tiene una clave compuesta, el orden de los elementos del registro definidos como claves debe coincidir con el orden de las columnas que son claves de la tabla de base de datos.

**defaultSelectCondition = #sqlCondition { condiciónSQL }**

Define parte del criterio de búsqueda de la cláusula WHERE de una sentencia SQL implícita. El valor de *defaultSelectCondition* no incluye la palabra clave SQL WHERE.

EGL proporciona una sentencia SQL implícita con una cláusula WHERE cuando se codifica una de estas sentencias EGL:

- **get**
- **open**
- **execute** (sólo al solicitar una sentencia SQL implícita DELETE o UPDATE)

Las sentencias SQL implícitas no se almacenan en el código fuente EGL. Para obtener una visión general de esas sentencias, consulte el apartado *Soporte SQL*.

*nivel*

Entero que indica la posición jerárquica de un campo de estructura. Si excluye este valor, el componente es un componente de registro; si incluye este valor, el componente es un componente de registro fijo.

*nombreCampoEstructura*

Nombre de un campo de estructura. Para conocer reglas, consulte la sección *Convenios de denominación*.

*tipoPrimitivo*

El tipo primitivo asignado al campo de estructura.

### *longitud*

La longitud del campo de estructura, que es un entero. El valor de un área de memoria basada en el elemento de estructura incluye el número especificado de caracteres o dígitos.

### *decimales*

Para un tipo numérico (BIN, DECIMAL, NUM, NUMC o PACF), puede especificar *decimales*, que es un entero que representa el número de posiciones después de la coma decimal. El número máximo de posiciones decimales es el menor de dos números: 18 o el número de dígitos declarado como *longitud*. La coma decimal no se almacena con los datos.

### *" máscaraFechaHora"*

Para elementos de tipo INTERVAL o TIMESTAMP, puede especificar *máscaraFechaHora*, que asigna un significado (como por ejemplo "dígito de año") a una posición dada en el valor de elemento. La máscara no se almacena con los datos.

### *nombreComponenteElementoDatos*

Especifica el nombre de un componente dataItem que actúa como modelo de formato del elemento de estructura que se declara. Para obtener detalles, consulte el apartado *typeDef*.

### **embed** *nombreComponenteRegistroSQL*

Especifica el nombre de un componente de registro de tipo sqlRecord e incorpora la estructura de dicho componente de registro en el registro actual. La estructura incorporada no añade un nivel de jerarquía al registro actual. Para obtener detalles, consulte el apartado *typeDef*.

### *propiedad*

Una propiedad de elemento, como se describe en el apartado *Visión general de las propiedades EGL y alteraciones temporales*. En un registro SQL, las propiedades de campo SQL son particularmente importantes.

## **Conceptos relacionados**

"Proyectos, paquetes y archivos EGL" en la página 13  
"Visión general de las propiedades de EGL" en la página 64  
"Componentes" en la página 17  
"Referencias a componentes" en la página 21  
"Componentes de registro" en la página 132  
"Soporte de SQL" en la página 229  
"Typedef" en la página 27

## **Tareas relacionadas**

"Diagrama de sintaxis para sentencias y mandatos EGL" en la página 755

## **Consulta relacionada**

"Matrices" en la página 74  
  
"Componente DataItem en formato fuente EGL" en la página 472  
"Formato fuente EGL" en la página 491  
"Componente de función en formato fuente EGL" en la página 527  
"Componente de registro indexado en formato fuente EGL" en la página 535  
"Componente de registro MQ en formato fuente EGL" en la página 662  
"Convenios de denominación" en la página 672  
"Tipos primitivos" en la página 34  
"Componente de programa en formato fuente EGL" en la página 728  
"Referencias a variables en EGL" en la página 58

“Componente de registro relativo en formato fuente EGL” en la página 740

“Componente de registro serie en formato fuente EGL” en la página 743

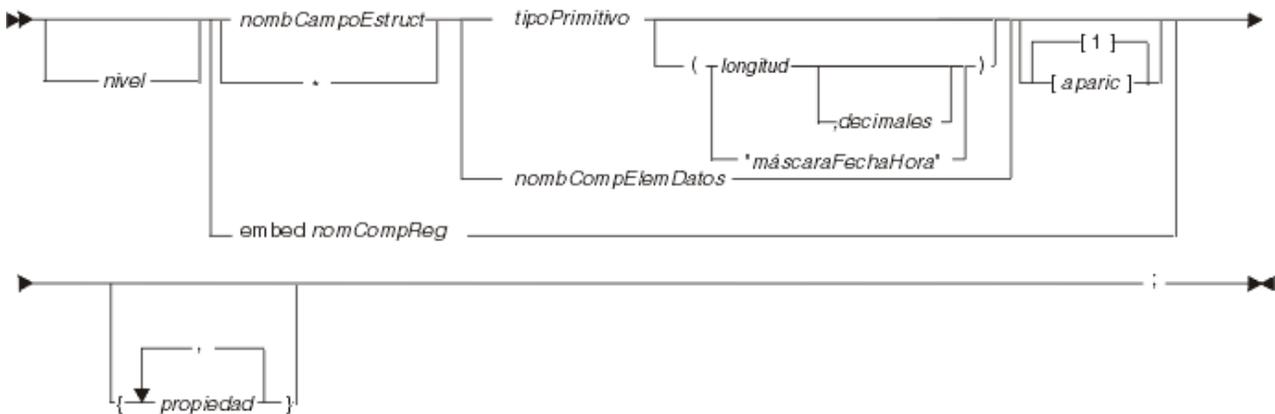
“Propiedades de elementos SQL” en la página 67

## Elemento de estructura en el formato fuente de EGL

A continuación se ofrece un ejemplo de un campo de estructura:

```
10 address;
20 street01 CHAR(20);
20 street02 CHAR(20);
```

El diagrama de sintaxis de un campo de estructura es el siguiente:



*nivel*

Entero que indica la posición jerárquica de un campo de estructura.

*nombreCampoEstructura*

Nombre de un campo de estructura. Para conocer las normas, consulte el apartado *Convenios de denominación*.

- \* Indica que el campo de estructura describe un *rellenador*, que es un área de memoria cuyo nombre no tiene importancia. No son válidos los asteriscos en las referencias a un área de memoria, como se indica en el apartado *Referencias a variables y constantes*.

*tipoPrimitivo*

El tipo primitivo asignado al campo de estructura.

*longitud*

La longitud del campo de estructura, que es un entero. El valor de un área de memoria basada en el campo de estructura incluye el número especificado de caracteres o dígitos.

*decimales*

Para un tipo numérico (BIN, DECIMAL, NUM, NUMC o PACF), puede especificar *decimals*, que es un entero que representa el número de posiciones después de la coma decimal. El número máximo de posiciones decimales es el menor de dos números: 18 o el número de dígitos declarado como *longitud*. La coma decimal no se almacena con los datos.

*" máscaraFechaHora "*

Para elementos de tipo INTERVAL o TIMESTAMP, puede especificar

*máscaraFechaHora*", que asigna un significado (como por ejemplo "dígito de año") a una posición dada en el valor de campo. La máscara no se almacena con los datos.

*nombreComponenteElementoDatos*

Especifica el nombre de un componente `dataItem` que actúa como modelo de formato del campo de estructura que se declara. Para obtener detalles, consulte el apartado *typeDef*.

**embed** *nombreComponenteRegistro*

Especifica el nombre de un componente de registro de registro e incorpora la estructura de dicho componente de registro en el registro actual. La estructura incorporada no añade un nivel de jerarquía al registro actual. Para obtener detalles, consulte el apartado *typeDef*.

*nombreComponenteRegistro*

Especifica el nombre de un componente de registro de registro e incluye la estructura de dicho componente de registro en el registro actual. En ausencia de la palabra *embed*, la estructura de registro se incluye como subestructura del campo de estructura que se declara. Para obtener detalles, consulte el apartado *typeDef*.

*apariciones*

El número de elementos de una matriz de elementos de estructura. El valor por omisión es 1, que indica que el campo de estructura no es una matriz, a menos que se especifique lo contrario. Para obtener detalles, consulte el apartado *Matrices*.

*propiedad*

Una propiedad de campo, como se describe en el apartado *Visión general de las propiedades EGL y alteraciones temporales*.

**Conceptos relacionados**

"Diagrama de sintaxis para funciones EGL" en la página 754

"Visión general de las propiedades de EGL" en la página 64

**Consulta relacionada**

"Matrices" en la página 74

"Convenios de denominación" en la página 672

"Tipos primitivos" en la página 34

"Referencias a variables en EGL" en la página 58

"Typedef" en la página 27

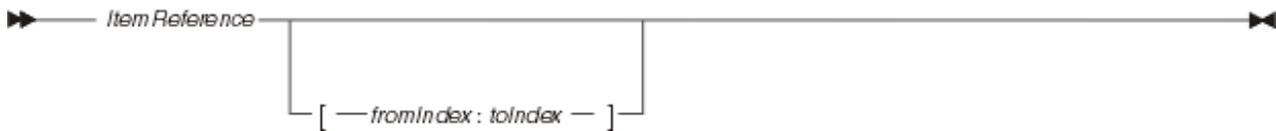
---

## Subseries

En cualquier contexto en el que haga referencia a un campo de caracteres, puede hacer referencia a una subserie que es un subconjunto secuencial de los caracteres de ese campo. Si un valor de campo es *ABCD*, puede hacer referencia a *BC* (por ejemplo), que son el segundo y tercer carácter.

Además, puede especificar una subserie en el lado izquierdo de una sentencia `assignment` si el campo destino es de tipo `CHAR`, `DBCHAR` o `UNICODE`. El área de subserie se rellena (con blancos, si es necesario) y el texto asignado no se extiende más allá del área de subserie (sino que se trunca, si es necesario).

La sintaxis de una referencia de subserie es la siguiente.



#### *itemReference*

A campo de carácter o HEXADECIMAL, pero no un literal. El elemento puede ser una variable del sistema o un elemento de matriz.

#### *fromIndex*

El primer carácter de interés del elemento, donde 1 representa el primer carácter del elemento de carácter, 2 representa el segundo, etc. Puede utilizar una expresión numérica que dé como resultado un entero, pero la expresión no puede incluir una invocación de función.

El valor de *fromIndex* representa una posición de byte a menos que *itemReference* haga referencia a un elemento de tipo DBCHAR o UNICODE, en cuyo caso el valor representa una posición de carácter de doble byte.

Cuenta a partir del carácter situado más a la izquierda, incluso aunque esté trabajando con un idioma bidireccional, como árabe o hebreo.

#### *toIndex*

El último carácter de interés del elemento, donde 1 representa el primer carácter del elemento de carácter, 2 representa el segundo, etc. Puede utilizar una expresión numérica que dé como resultado un entero, pero la expresión no puede incluir una invocación de función.

El valor de *toIndex* representa una posición de byte a menos que *itemReference* haga referencia a un elemento de tipo DBCHAR o UNICODE, en cuyo caso el valor representa una posición de carácter de doble byte.

Cuenta a partir del carácter situado más a la izquierda, incluso aunque esté trabajando con un idioma bidireccional, como árabe o hebreo.

#### **Conceptos relacionados**

“Referencias a variables en EGL” en la página 58

#### **Tareas relacionadas**

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

#### **Consulta relacionada**

“Expresiones numéricas” en la página 504

---

## **Diagrama de sintaxis para funciones EGL**

En el tema que describe una función de sistema EGL determinada, un diagrama de sintaxis ofrece detalles acerca del tipo de cada parámetro de función y del tipo de valor devuelto, si existe. El nombre de la biblioteca de funciones se especifica anteriormente en el tema.

A continuación se ofrece un ejemplo de diagrama:

```
StrLib.clip(texto STRING in)
returns (resultado STRING)
```

El diagrama se inicia con el nombre de la función y muestra una lista de especificaciones de parámetro, cada una de las cuales incluye los siguientes detalles:

- El nombre del parámetro, que puede o no especificarse; en este ejemplo, el nombre del parámetro uno es *texto*.
- El tipo del parámetro, que es un tipo del lenguaje EGL o una combinación de tipos. (Si el tipo no está en el lenguaje EGL, se suministra una descripción más detallada en el tema). En este ejemplo, el tipo es STRING.
- El modificador **in**, **out** o **inOut**, según se describe en los *Parámetros de función*.

Si la especificación de parámetro está entre corchetes ([ ]), el argumento asociado con ese parámetro es opcional. Si la especificación está entre llaves ({ }), el argumento también es opcional, pero en este caso puede incluir varios argumentos del mismo tipo.

Si la función devuelve un valor, el diagrama muestra la palabra *Returns* y un nombre y un tipo entre paréntesis. El tema hace referencia a ese nombre al describir el valor de retorno, pero por lo demás carece de significado.

Si una cláusula *returns* está entre corchetes ([ ]), el valor de retorno es opcional.

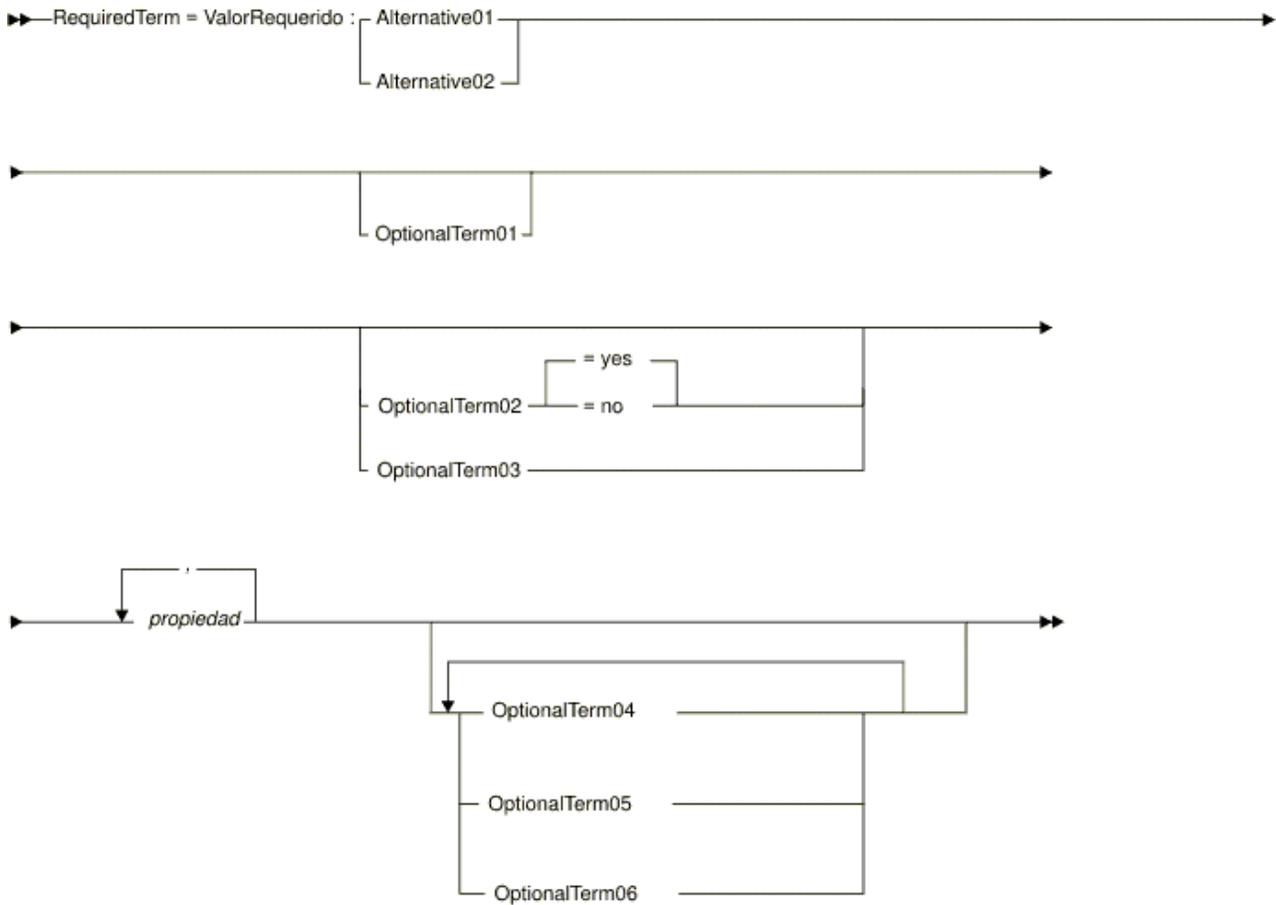
#### **Consulta relacionada**

“Invocaciones de función” en la página 518  
 “Parámetros de función” en la página 522  
 “Biblioteca ConsoleLib de EGL” en la página 757  
 “Biblioteca J2EELib de EGL” en la página 802  
 “Biblioteca JavaLib de EGL” en la página 805  
 “Biblioteca LobLib de EGL” en la página 831  
 “Biblioteca MathLib de EGL” en la página 839  
 “Biblioteca ReportLib de EGL” en la página 861  
 “Biblioteca StrLib de EGL” en la página 867  
 “Biblioteca SysLib de EGL” en la página 887  
 “Biblioteca VGLib de EGL” en la página 916

---

## **Diagrama de sintaxis para sentencias y mandatos EGL**

El diagrama de sintaxis de IBM le permite ver rápidamente cómo construir un mandato de construcción o una sentencia EGL. Este es un ejemplo de un diagrama de este tipo:



Lea el diagrama de izquierda a derecha y de arriba a abajo, siguiendo la *ruta principal*, que es la línea que empieza a la izquierda con doble flecha (>>). A medida que sigue la ruta principal, puede seleccionar una entrada en una ruta subordinada, en cuyo caso continuará leyendo de izquierda a derecha por la ruta subordinada.

En el ejemplo, la ruta principal consta de cuatro segmentos de línea. Es importante fijarse en esto. El segundo y tercer segmentos de línea de la ruta principal empieza con una sola flecha (>) e incluye información subordinada. El cuarto segmento de línea de la línea de ruta principal también empieza con una sola flecha (>), incluye flechas de retorno e información subordinada, y finaliza con dos flechas enfrentadas (><).

Un término (o símbolo) que no esté en cursiva debe especificarse exactamente como se muestra. En el ejemplo, especifique el término **RequiredTerm** tal cual. Por el contrario, un término en cursiva es un espacio reservado para un valor que especifique. En el ejemplo podría incluir cualquiera de los símbolos siguientes en lugar de *RequiredValue*:

```
myVariable
50
"0h!"
```

Los requisitos específicos para un término en cursiva (por ejemplo, si es adecuada una serie o un número) se explican en el texto que sigue al diagrama de sintaxis, no en el propio diagrama de sintaxis.

Si un diagrama muestra un carácter no alfanumérico, teclee ese carácter como parte de la sintaxis. Tras especificar un valor para *RequiredValue*, por ejemplo, teclee dos puntos (:) y un espacio en blanco.

Si se le permite seleccionar entre varios términos, los términos se muestran en una pila. En el ejemplo, puede especificar el término **Alternative01** o **Alternative02**.

Si (como en este caso) *debe seleccionar* un término de los listados en una pila, una de las opciones (especificada de forma arbitraria) está en la línea superior de la pila. Si no se le solicita que seleccione un término, todos los términos están debajo de la línea superior de la pila, como sucede en **OptionalTerm01**.

Un valor que está en una ruta pero que se muestra de manera elevada (como sucede con = **yes**) es el valor por omisión para la pila en que aparece el valor. El ejemplo indica que puede especificar cualquiera de las siguientes series y las dos primeras son equivalentes:

optionalTerm01 = yes

optionalTerm01

optionalTerm01 = no

OptionalTerm02

Una flecha de retorno a la izquierda encima de un término indica que puede utilizar el término repetidamente. En el ejemplo, puede especificar valores para *propiedad*, cada uno de ellos separado del siguiente por una coma.

Una flecha de retorno a la izquierda encima de una pila vertical significa que puede elegir de la lista de entradas en cualquier orden. En el ejemplo, cada una de las siguientes series es válida (al igual que lo son otras variaciones), pero ninguna es necesaria:

OptionalTerm04 OptionalTerm05

OptionalTerm06

OptionalTerm04 OptionalTerm06 OptionalTerm05

---

## Bibliotecas del sistema

### Biblioteca ConsoleLib de EGL

La biblioteca de consola proporciona la funcionalidad de consoleUI para los programas EGL. La utilización del calificador **ConsoleLib** (por ejemplo, **ConsoleLib.activateWindow**) es opcional.

Función	Descripción
activateWindow ( <i>ventana</i> )	Convierte la ventana especificada en la ventana activa y actualiza la variable <i>activeWindow</i> de <b>ConsoleLib</b> en consecuencia.
activateWindowByName ( <i>nombre</i> )	Convierte la ventana especificada en la ventana activa y actualiza la variable <i>activeWindow</i> de <b>ConsoleLib</b> en consecuencia.

Función	Descripción
cancelArrayDelete ()	Finaliza la operación <i>delete</i> actual en proceso durante la ejecución de un bloque de código de evento <b>BEFORE_DELETE</b> <b>OpenUI</b> .
cancelArrayInsert ()	Finaliza la operación <i>insert</i> actual en proceso durante la ejecución de un bloque de código de evento <b>BEFORE_INSERT</b> <b>OpenUI</b> .
clearActiveForm ()	Borra los almacenamientos intermedios de visualización de todos los campos.
clearActiveWindow ()	Elimina todo el material visualizado de la ventana activa.
clearFields ([ <i>campoConsola</i> {, <i>campoConsola</i> }]})	Borra los almacenamientos intermedios de visualización de los campos especificados en el formulario activo. Si no se han especificado campos, se borran todos los campos del formulario.
clearFieldsByName ( <i>nombreCampo</i> {, <i>nombreCampo</i> })	Borra los almacenamientos intermedios de visualización de los campos nombrados en el formulario activo. Si no se han nombrado campos, se borran todos los campos del formulario.
clearForm ( <i>formularioConsola</i> )	Borra los almacenamientos intermedios de visualización de todos los campos.
clearWindow ( <i>ventana</i> )	Elimina todo el material visualizado de la ventana especificada.
clearWindowByName ( <i>nombre</i> )	Elimina todo el material visualizado de la ventana especificada.
closeActiveWindow ()	Borra la ventana de la pantalla, libera los recursos asociados a esa ventana y activa la ventana que estaba activa anteriormente.
closeWindow ( <i>ventana</i> )	Borra la ventana de la pantalla, libera los recursos asociados a esa ventana y activa la ventana que estaba activa anteriormente.
closeWindowByName ( <i>nombre</i> )	Borra la ventana de la pantalla, libera los recursos asociados a esa ventana y activa la ventana que estaba activa anteriormente.
<i>result</i> = currentArrayCount ()	Devuelve el número de elementos de la matriz dinámica que está asociada al formulario activo actual
<i>result</i> = currentArrayDataLine ()	Devuelve el número del registro del programa dentro de la matriz de programas que se visualiza en la línea actual de una matriz de pantalla durante o inmediatamente después de la sentencia <b>OpenUI</b> .

<b>Función</b>	<b>Descripción</b>
<i>result</i> = <i>currentArrayScreenLine</i> ()	Devuelve el número del registro de pantalla actual en la matriz de pantalla correspondiente durante una sentencia <b>OpenUI</b> .
<i>displayAtLine</i> ( <i>texto</i> , <i>línea</i> )	Visualiza una serie en un lugar especificado dentro de la ventana activa.
<i>displayAtPosition</i> ( <i>texto</i> , <i>línea</i> , <i>columna</i> )	Visualiza una serie en un lugar especificado dentro de la ventana activa.
<i>displayError</i> ( <i>msj</i> )	Provoca la creación y visualización de una ventana de error y visualiza el mensaje de error en esa ventana.
<i>displayFields</i> ([ <i>campoConsola</i> {, <i>campoConsola</i> }]])	Visualiza valores de campo de formulario en la consola.
<i>displayFieldsByName</i> ( <i>nombreCampoConsola</i> {, <i>nombreCampoConsola</i> )	Visualiza valores de campo de formulario en la consola.
<i>displayForm</i> ( <i>formularioConsola</i> )	Visualiza el formulario en la ventana activa.
<i>displayFormByName</i> ( <i>nombreFormulario</i> )	Visualiza el formulario en la ventana activa.
<i>displayLineMode</i> ( <i>texto</i> )	Visualiza una serie en <i>modalidad de línea</i> en lugar de en <i>modalidad de formulario/ventana</i> .
<i>displayMessage</i> ( <i>msj</i> )	Visualiza una serie en un lugar especificado dentro de la ventana activa y utiliza los valores <i>messageLine</i> de la ventana activa para identificar dónde desea visualizar la serie.
<i>drawBox</i> ( <i>fila</i> , <i>columna</i> , <i>profundidad</i> , <i>anchura</i> )	Traza un rectángulo en la ventana activa con la ubicación y las dimensiones especificadas.
<i>drawBoxWithColor</i> ( <i>fila</i> , <i>columna</i> , <i>profundidad</i> , <i>anchura</i> , <i>Color</i> )	Traza un rectángulo en la ventana activa con la ubicación, las dimensiones y el color especificados.
<i>result</i> = <i>getKey</i> ()	Lee una tecla de la entrada y devuelve el código de entero para la tecla.
<i>result</i> = <i>getKeyCode</i> ( <i>keyName</i> )	Devuelve el código de entero de tecla de la tecla nombrada en la serie.
<i>result</i> = <i>getKeyName</i> ( <i>keyCode</i> )	Devuelve el nombre que representa el código de tecla entero.
<i>gotoField</i> ( <i>campoConsola</i> )	Mueve el cursor al campo de formulario especificado.
<i>gotoFieldByName</i> ( <i>nombre</i> )	Mueve el cursor al campo de formulario especificado.
<i>gotoMenuItem</i> ( <i>elemento</i> )	Mueve el cursor del menú al elemento de menú especificado.
<i>gotoMenuItemByName</i> ( <i>nombre</i> )	Mueve el cursor del menú al elemento de menú especificado.
<i>hideAllMenuItems</i> ()	Oculto todos los elementos de menú del menú visualizado actualmente.
<i>hideErrorWindow</i> ()	Oculto la ventana de error.

Función	Descripción
hideMenuItem ( <i>elemento</i> )	Oculto un elemento de menú especificado de forma que un usuario no pueda seleccionarlo.
hideMenuItemByName ( <i>nombre</i> )	Oculto un elemento de menú especificado de forma que un usuario no pueda seleccionarlo.
<i>result</i> = isCurrentField ( <i>consoleField</i> )	Devuelve <b>true</b> si el cursor está en el campo de formulario especificado; en caso contrario, devuelve <b>false</b> .
<i>result</i> = isCurrentFieldByName ( <i>name</i> )	Devuelve <b>true</b> si el cursor está en el campo de formulario especificado; en caso contrario, devuelve <b>false</b> .
<i>result</i> = isFieldModified ( <i>consoleField</i> )	Devuelve <b>true</b> si el usuario ha cambiado el contenido del campo de formulario especificado; un valor <b>false</b> indica que el campo no se ha editado.
<i>result</i> = isFieldModifiedByName ( <i>name</i> )	Devuelve <b>true</b> si el usuario ha cambiado el contenido del campo de formulario especificado; un valor <b>false</b> indica que el campo no se ha editado.
<i>result</i> = lastKeyTyped ()	Devuelve el código entero de la última clave física que se pulsó en el teclado.
nextField ()	Mueve el cursor al campo de formulario siguiente según el orden de desplazamiento de campos definido.
openWindow ( <i>ventana</i> )	Hace visible una ventana y la añade a la parte superior de la pila de ventanas. El formulario se visualiza en la ventana.
openWindowByName ( <i>nombre</i> )	Hace visible una ventana y la añade a la parte superior de la pila de ventanas.
openWindowWithForm ( <i>Ventana, formulario</i> )	Hace visible una ventana y la añade a la parte superior de la pila de ventanas. El tamaño de la ventana cambiará para albergar el formulario especificado si no se definió el tamaño de la ventana cuando esta se declaró.
openWindowWithFormByName ( <i>nombreVentana, nombreFormulario</i> )	Hace visible una ventana y la añade a la parte superior de la pila de ventanas.
previousField ()	Mueve el cursor al campo de formulario anterior según el orden de desplazamiento de campos definido.
<i>result</i> = promptLineMode ( <i>prompt</i> )	Muestra un mensaje de solicitud al usuario en un entorno de <i>modalidad de línea</i> .
scrollDownLines ( <i>númeroLíneas</i> )	Desplaza la tabla de datos hacia el inicio de los datos. (Es decir, índices de registro más pequeños)
scrollDownPage ()	Desplaza la tabla de datos hacia el inicio de los datos. (Es decir, índices de registro más pequeños)

Función	Descripción
scrollUpLines ( <i>númeroLíneas</i> )	Desplaza la tabla de datos hacia el final de los datos. (Es decir, índices de registro más grandes)
scrollUpPage ()	Desplaza la tabla de datos hacia el final de los datos (es decir, índices de registro más grandes)
setArrayLine ( <i>númeroRegistro</i> )	Mueve la selección al registro de programa especificado. La tabla de datos se desplaza en la pantalla si es necesario para hacer visible el registro seleccionado.
setCurrentArrayCount ( <i>cuenta</i> )	Establece cuántos registros existen en la matriz de programa. Debe llamarse antes de la sentencia <b>OpenUI</b> .
showAllMenuItems ()	Muestra todos los elementos de menú para la selección de usuario.
showHelp ( <i>teclaAyuda</i> )	Visualiza la pantalla de ayuda de <b>ConsoleUI</b> durante la ejecución del programa EGL.
showMenuItem ( <i>elemento</i> )	Muestra el elemento de menú especificado para la selección de usuario.
showMenuItemByName( <i>nombre</i> )	Muestra el elemento de menú especificado para la selección de usuario.

Variables	Descripción
activeForm	El formulario visualizado más recientemente en la ventana activa.
activeWindow	La ventana situada en la parte superior; es el destino de las operaciones de ventana cuando no se especifica ningún nombre de ventana.
commentLine	La línea de ventana en la que se visualizan mensajes de comentario.
CurrentDisplayAttrs	Valores aplicados a los elementos visualizados mediante las funciones de visualización.
currentRowAttrs	Atributos de resaltado aplicados a la fila actual.
cursorWrap	Si el valor es <b>true</b> , el cursor pasa al primer campo del formulario; si el valor es <b>false</b> , la sentencia finaliza cuando el cursor se mueve hacia adelante desde el último campo de entrada del formulario.
defaultDisplayAttributes	Valores predeterminados de los <i>atributos de presentación</i> para objetos nuevos.
defaultInputAttributes	Los valores predeterminados de los <i>atributos de presentación</i> para operaciones de entrada.

Variables	Descripción
deferInterrupt	Si el valor es <b>true</b> , el programa captura señales <b>INTR</b> y las anota en la variable <i>interruptRequested</i> cuya supervisión es entonces responsabilidad del programa. En Windows, la señal se simula cuando se pulsa la tecla lógica <b>INTERRUPT</b> que por omisión es <b>CONTROL_C</b> .
deferQuit	Si el valor es <b>true</b> , el programa captura señales <b>QUIT</b> y las anota en la variable <i>interruptRequested</i> cuya supervisión es entonces responsabilidad del programa. En Windows, la señal se simula cuando se pulsa la tecla lógica <b>QUIT</b> que por omisión es <b>CONTROL_\<b>\</b></b> .
definedFieldOrder	Si el valor es <b>true</b> , las teclas de flecha arriba y abajo se mueven a los campos anterior y siguiente según el orden de travesía. Si el valor es <b>false</b> , arriba y abajo se mueven al campo que hay en esa dirección física en la pantalla.
errorLine	La ventana en la que se visualizan los mensajes de error.
errorWindow	La ubicación de la ventana en la que se visualizan los mensajes de error en la pantalla ConsoleUI.
errorWindowVisible	Si el valor es <b>true</b> , la ventana de error se está visualizando actualmente en la pantalla
formLine	La línea de la ventana en la que se visualizan los formularios.
interruptRequested	Esto indica que se ha recibido (o simulado) una señal <b>INTR</b> .
key_accept	Tecla para la finalización satisfactoria de las sentencias <b>OpenUI</b> . La tecla predeterminada es <b>ESCAPE</b> .
key_deleteLine	Tecla para suprimir la fila actual de una matriz de pantalla. La tecla predeterminada es <b>F2</b> .
key_help	Tecla para mostrar la ayuda sensible al contexto durante las sentencias <b>OpenUI</b> . La tecla predeterminada es <b>CTRL_W</b> .
key_insertLine	Tecla para insertar una fila en una matriz de pantalla. La tecla predeterminada es <b>F1</b> .
key_interrupt	Tecla para simular una señal <b>INTR</b> . La tecla predeterminada es <b>CTRL_C</b> .
key_pageDown	Tecla para pasar página hacia delante en una matriz de pantalla (tabla de datos). La tecla predeterminada es <b>F3</b> .
key_pageUp	Tecla para pasar página hacia atrás en una matriz de pantalla (tabla de datos). La tecla predeterminada es <b>F4</b> .

Variables	Descripción
key_quit	Tecla para simular una señal QUIT. La tecla predeterminada es <b>CTRL_\ </b>
menuLine	La línea de la ventana en la que se visualizan los menús.
messageLine	La línea de la ventana en la que se visualizan los mensajes.
messageResource	El nombre de archivo del paquete compuesto de recursos.
promptLine	La línea de la ventana en la que se visualizan los mensajes de error.
quitRequested	Indica que se ha recibido (o simulado) una señal <b>QUIT</b> .
screen	Ventana definida automáticamente, de forma predeterminada y sin bordes; las dimensiones coinciden con la superficie de visualización disponible.
sqlInterrupt	Si el valor es <b>true</b> , el usuario puede interrumpir las sentencias SQL que se procesan. Si el valor es <b>false</b> , el usuario solo puede interrumpir las sentencias <b>OpenUI</b> . Se utiliza en combinación con las variables <i>deferInterrupt</i> y <i>deferQuit</i> .

## activeForm

La variable de sistema **ConsoleLib.activeForm** es el formulario visualizado más recientemente en la ventana activa.

Tipo: ConsoleForm

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## activateWindow()

La función de sistema **ConsoleLib.activateWindow** convierte la ventana especificada en la ventana activa y actualiza la variable *activeWindow*.

```
ConsoleLib.activateWindow(ventana1 Window inOut)
```

*ventana1*

La ventana a activar.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## **activeWindow**

La variable de sistema **ConsoleLib.activeWindow** es la ventana situada en la parte superior o la que se ha activado más recientemente. **ConsoleLib.activeWindow** es el destino de las operaciones de ventana cuando no se especifica ningún nombre de ventana.

Tipo: Window

### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## **activateWindowByName()**

La función del sistema **ConsoleLib.activateWindowByName** convierte la ventana especificada en la ventana activa y actualiza la variable *activeWindow* de **ConsoleLib** en consecuencia.

```
ConsoleLib.activateWindowByName(nombre STRING in)
```

*nombre*

El nombre de la ventana.

### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## **cancelArrayDelete()**

La función del sistema **ConsoleLib.cancelArrayDelete** finaliza la operación *delete* actual en proceso durante la ejecución de un bloque de código de evento **BEFORE\_DELETE OpenUI**.

Si en tiempo de ejecución, esta función se ejecuta fuera del ámbito de una sentencia **OpenUI**, el efecto es una operación nula.

```
ConsoleLib.cancelArrayDelete()
```

### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## **cancelArrayInsert()**

La función del sistema **ConsoleLib.cancelArrayInsert** finaliza la operación *insert* actual en proceso durante la ejecución de un bloque de código de evento **BEFORE\_INSERT OpenUI**. Si en tiempo de ejecución, esta función se ejecuta fuera del ámbito de una sentencia **OpenUI**, el efecto es una operación nula.

```
ConsoleLib.cancelArrayInsert()
```

### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## clearActiveForm()

La función del sistema **ConsoleLib.clearActiveForm** borra los almacenamientos intermedios de visualización de todos los campos. Esta función no tiene efecto sobre los elementos de datos enlazados; los datos almacenados en los elementos de datos enlazados no se borran.

```
ConsoleLib.clearActiveForm()
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## clearActiveWindow

La función del sistema **ConsoleLib.clearActiveWindow** elimina todo el material visualizado de la ventana activa. Esto incluye la eliminación de la información constante visualizada en el formulario actual. Si la ventana activa tiene un borde, el borde no se elimina. La sentencia no afecta al orden de la pila de ventanas ni a las ventanas que están por encima de ella en la pila.

```
ConsoleLib.clearActiveWindow()
```

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## clearFields()

La función del sistema **ConsoleLib.clearFields** borra los almacenamientos intermedios de visualización de los campos especificados. Si no se han especificado campos, se borran todos los campos. Esta función no tiene efecto sobre los elementos de datos enlazados; los datos almacenados en los elementos de datos enlazados no se borran.

```
ConsoleLib.clearFields(
 [campoConsola1 ConsoleField inOut
 { , campoConsola1 ConsoleField inOut}
])
```

*campoConsola1*

El nombre de la variable de tipo ConsoleField.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## clearFieldsByName()

La función de sistema **ConsoleLib.clearFieldsByName** borra los campos especificados en pantalla; y borra todos los campos si no hay ningún campo especificado. Las variables enlazadas a los campos en pantalla no se ven afectadas.

```
ConsoleLib.clearFieldsByName(
 [nombreCampo STRING in
 { , nombreCampo STRING in}])
```

*nombreCampo*

El valor de un campo Nombre de ConsoleField.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## clearForm()

La función del sistema **ConsoleLib.clearForm** borra todos los campos del formulario especificado. Las variables enlazadas a dichos campos no se ven afectadas.

```
ConsoleLib.clearForm(formularioConsola ConsoleForm inOut)
```

*formularioConsola*

Una variable de tipo ConsoleForm.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## clearWindow()

La función del sistema **ConsoleLib.clearWindow** elimina todo el material visualizado de la ventana especificada. Esto incluye la eliminación de la información constante visualizada en el formulario actual. Si la ventana tiene un borde, el borde no se elimina. La sentencia no afecta al orden de la pila de ventanas ni a las ventanas que están por encima de ella en la pila.

```
ConsoleLib.clearWindow(ventana1 Window inOut)
```

*ventana1*

La ventana a borrar.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## clearWindowByName()

La función del sistema **ConsoleLib.clearWindowByName** elimina todo el material visualizado de la ventana especificada. Esto incluye la eliminación de la información constante visualizada en el formulario actual. Si la ventana tiene un borde, el borde no se elimina. La sentencia no afecta al orden de la pila de ventanas. La variable **ActiveWindow** hace referencia a la ventana situada en la parte superior de la pila de visualización.

```
ConsoleLib.cleaWindowByName(nombre STRING in)
```

*nombre*

El nombre de la ventana.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## **closeActiveWindow()**

La función del sistema **ConsoleLib.closeActiveWindow** borra la ventana de la pantalla, libera los recursos asociados a la ventana borrada y activa la ventana que estaba activa anteriormente.

Después de invocar **ConsoleLib.closeActiveWindow**, la ventana no puede volver a abrirse mediante **ConsoleLib.openWindow** ni **ConsoleLib.openWindowByName**. Además, no puede cerrar la ventana SCREEN.

**ConsoleLib.closeActiveWindow( )**

### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## **closeWindow()**

La función de biblioteca de consola **ConsoleLib.closeWindow** borra la ventana especificada de la pantalla, libera los recursos asociados a la ventana borrada y activa la ventana que estaba activa anteriormente.

Después de invocar **ConsoleLib.closeWindow**, la ventana no puede volver a abrirse mediante **ConsoleLib.openWindow** ni **ConsoleLib.openWindowByName**. Además, no puede cerrar la ventana SCREEN.

**ConsoleLib.closeWindow(ventana1 Window inOut)**

*ventana1*

El objeto de ventana especificado en la pantalla.

### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## **closeWindowByName()**

La función del sistema **ConsoleLib.closeWindowByName** borra de la pantalla la ventana nombrada, libera los recursos asociados a la ventana cerrada y activa la ventana que estaba activa anteriormente.

Después de invocar **ConsoleLib.closeWindowByName**, la ventana no puede volver a abrirse mediante **ConsoleLib.openWindow** ni **ConsoleLib.openWindowByName**. La ventana de consola permanece abierta.

**ConsoleLib.closeWindowByName(nombre STRING in)**

*nombre*

El nombre de la ventana.

### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## **commentLine**

La variable de sistema **ConsoleLib.commentLine** es la línea de ventana en la que se visualizan mensajes de comentario.

Tipo: Integer

### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## **currentArrayCount()**

La función de sistema **ConsoleLib.currentArrayCount** devuelve un número de elementos de la matriz dinámica que está asociada al formulario activo actual.

Se recomienda evitar el uso de esta función, que se utiliza para ayudar a migrar aplicaciones que se han escrito con Informix 4GL. En su lugar, utilice la función específica de la matriz **getSize**, tal como se describe en el apartado *Matrices*.

```
ConsoleLib.currentArrayCount()
returns (resultado INT)
```

*resultado*

El número de elementos.

### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

### **Consulta relacionada**

“Matrices” en la página 74

“Biblioteca ConsoleLib de EGL” en la página 757

## **currentArrayDataLine()**

La función del sistema **ConsoleLib.currentArrayDataLine** devuelve el número del registro del programa dentro de la matriz de programas que se visualiza en la línea actual de una matriz de pantalla durante o inmediatamente después de la sentencia **openUI**.

```
ConsoleLib.currentArrayDataLine()
returns (resultado INT)
```

*resultado*

Cualquier entero.

### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## **currentArrayScreenLine()**

La función del sistema **ConsoleLib.currentArrayScreenLine** devuelve el número del registro de pantalla actual en la matriz de pantalla correspondiente durante una sentencia **openUI**.

```
ConsoleLib.currentArrayScreenLine()
returns (resultado INT)
```

*resultado*

Cualquier entero.

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

### **currentDisplayAttrs**

La variable de sistema **ConsoleLib.currentDisplayAttrs** especifica las características de visualización de cualquier texto que se mostrará después de que se haya establecido la variable.

Las variables de tipo *PresentationAttributes* incluyen los campos *color*, *intensity* y *highlight*. Para obtener detalles, consulte el apartado *Componentes de ConsoleUI y variables relacionadas*.

Tipo: *PresentationAttributes*

#### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

“Componentes de ConsoleUI y variables relacionadas” en la página 180

### **currentRowAttrs**

La variable de sistema **ConsoleLib.currentRowAttrs** son atributos de resultado aplicados a la fila actual de una matriz de pantalla.

Las variables de tipo *PresentationAttributes* incluyen los campos *color*, *intensity* y *highlight*. Para obtener detalles, consulte el apartado *Componentes de ConsoleUI y variables relacionadas*.

Tipo: *PresentationAttributes*

#### **Consulta relacionada**

**currentDisplayAttrs**

“Biblioteca ConsoleLib de EGL” en la página 757

### **cursorWrap**

La variable de sistema **ConsoleLib.cursorWrap** indica si el cursor pasa al primer campo del formulario. Esta función devuelve **true** si el cursor pasa al primer campo del formulario y **false** si la sentencia finaliza cuando el cursor se mueve hacia adelante desde el último campo de entrada del formulario.

Tipo: Boolean

#### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

### **defaultDisplayAttributes**

La variable de sistema **ConsoleLib.defaultDisplayAttributes** contiene los valores utilizados para *PresentationAttributes* en las variables.

Las variables de tipo `PresentationAttributes` incluyen los campos `color`, `intensity` y `highlight`. Para obtener detalles, consulte el apartado *Componentes de ConsoleUI y variables relacionadas*.

Tipo: `PresentationAttributes`

#### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

### **defaultInputAttributes**

La variable de sistema `ConsoleLib.defaultInputAttributes` contiene los valores predeterminados de los atributos de presentación para las operaciones de entrada.

Las variables de tipo `PresentationAttributes` incluyen los campos `color`, `intensity` y `highlight`. Para obtener detalles, consulte el apartado *Componentes de ConsoleUI y variables relacionadas*.

Tipo: `PresentationAttributes`

#### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

### **deferInterrupt**

La variable de biblioteca de Interfaz de usuario de consola `ConsoleLib.deferInterrupt` identifica el comportamiento de la aplicación cuando recibe la señal `INTERRUPT`. Si los resultados son `true`, el programa captura señales `INTR` y las anota en la variable `interruptRequested` cuya supervisión es entonces responsabilidad del programa. En Windows, la señal se simula cuando se pulsa la tecla lógica `INTERRUPT` que por omisión es `CONTROL_C`. Si los resultados son `false`, el programa finaliza cuando se pulsa la tecla `interrumpir`.

Tipo: `Boolean`

#### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

### **deferQuit**

Para la variable de sistema `ConsoleLib.deferQuit`, si el valor es `true`, el programa captura señales `QUIT` y las anota en la variable `quitRequested` cuya supervisión es entonces responsabilidad del programa. En Windows, la señal se simula cuando se pulsa la tecla lógica `QUIT` que por omisión es `CONTROL_`. Si el valor es `false`, al recibir una señal de abandono finalizará la aplicación.

Tipo: `Boolean`

#### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

### **definedFieldOrder**

La variable de Interfaz de usuario de consola `ConsoleLib.definedFieldOrder` determina el comportamiento de las teclas de flecha arriba/abajo al efectuar una entrada en un formulario. Si el valor es `true`, el cursor recorre los campos según el

orden de definición al utilizar las teclas de flecha arriba/abajo. Si el valor es **false**, el cursor sube y baja según la disposición física de los campos de la pantalla.

Tipo: Boolean

#### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

### displayAtLine()

La función del sistema **ConsoleLib.displayAtLine** visualiza una serie en un lugar especificado dentro de la ventana activa.

```
ConsoleLib.displayAtLine(
 texto STRING in,
 línea INT in)
```

*texto*

La serie a visualizar.

*línea*

El número de la línea en la que se va a visualizar la serie.

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

#### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

### displayAtPosition()

La función del sistema **ConsoleLib.displayAtPosition** visualiza una serie en un lugar especificado dentro de la ventana activa.

```
ConsoleLib.displayAtPosition(
 texto STRING in,
 línea INT in,
 columna INT in)
```

*texto*

La serie a visualizar.

*línea*

El número de la línea en la que se va a visualizar la serie.

*columna*

El número de la columna en la que se va a visualizar la serie.

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

#### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

### displayError()

La función del sistema **ConsoleLib.displayError** provoca la creación y visualización de una ventana de error y visualiza el mensaje de error en esa ventana. La ventana de error flota sobre el resto de ventanas hasta que se cierra llamando a *hideErrorWindow()* o cuando se pulsa una tecla. Si procede, se activará la campana de terminal.

```
ConsoleLib.displayError(msj STRING in)
```

*msj*

El mensaje de error a visualizar.

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

#### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

### displayFields()

La función del sistema **ConsoleLib.displayFields** visualiza en la consola los valores de campo de formulario. Si los elementos de datos están enlazados a los campos, se recuperarán los datos de esos elementos y se formatearán según las reglas especificadas con el campo de formulario. Para un campo de formulario no enlazado, los datos pueden establecerse directamente en los campos accediendo al campo **ConsoleField.value**.

```
ConsoleLib.displayFields(
 [campoConsola1 ConsoleField in
 {, campoConsola1 ConsoleField in}
]
)
```

*campoConsola1*

El nombre de la variable de tipo ConsoleField.

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

#### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

### displayFieldsByName()

La función del sistema **ConsoleLib.displayFieldsByName** visualiza en la consola los valores de campo de formulario. Si los elementos de datos están enlazados a los campos, se recuperarán los datos de esos elementos y se formatearán según las reglas especificadas con el campo de formulario. Para un campo de formulario no enlazado, los datos pueden establecerse directamente en estos campos accediendo al campo **ConsoleField.value**.

```
ConsoleLib.displayFieldsByName(
 nombreCampoConsola1 ConsoleFieldName in
 {, nombreCampoConsola1 ConsoleFieldName in}
)
```

*nombreCampoConsola1*

Los nombres de los campos a visualizar.

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

#### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

### displayForm()

La función del sistema **ConsoleLib.displayForm** visualiza el formulario especificado en la ventana activa.

```
ConsoleLib.displayForm(formularioConsola ConsoleForm in)
```

*formularioConsola*

El nombre de la variable de tipo ConsoleForm.

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

### **displayFormByName()**

La función de sistema **ConsoleLib.displayFormByName** visualiza el formulario nombrado en la ventana activa.

**ConsoleLib.displayFormByName**(*nombreFormulario* **STRING** in)

*nombreFormulario*

El valor del campo Nombre de ConsoleForm.

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

### **displayLineMode()**

La función del sistema **ConsoleLib.displayLineMode** visualiza la serie designada en **modalidad de línea** en lugar de en **modalidad de formulario/ventana**. El valor de la serie se envía a la ubicación de *salida* estándar del sistema en ejecución. Todas las características de visualización como por ejemplo la acomodación y el desplazamiento quedan bajo la responsabilidad de la interfaz de salida estándar.

**ConsoleLib.displayLine**(*texto* **STRING** in)

*texto*

La serie a visualizar.

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

### **displayMessage()**

La función del sistema **ConsoleLib.displayMessage** muestra una serie en la línea de mensajes de la ventana activa. La función utiliza los valores *MessageLine* de la ventana activa para saber dónde visualizar la serie.

**ConsoleLib.displayMessage**(*msj* **STRING** in)

*msj*

El mensaje a visualizar.

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## drawBox()

La función del sistema **ConsoleLib.drawBox** traza un rectángulo en la ventana activa con la esquina superior izquierda en *fila*, *columna* para los dos primeros enteros y con *profundidad*, *anchura* para los dos enteros siguientes. La fila y la columna son relativas a la esquina superior izquierda de la ventana actual.

```
ConsoleLib.drawBox(
 fila INT in,
 columna INT in,
 profundidad INT in,
 anchura INT in)
```

*fila*

El número de fila en relación con la esquina superior izquierda de la ventana.

*columna*

El número de columna en relación con la esquina superior izquierda de la ventana.

*profundidad*

La profundidad o altura del recuadro.

*anchura*

La anchura del recuadro.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## drawBoxWithColor()

La función de sistema **ConsoleLib.drawBoxWithColor** traza un rectángulo en la ventana activa con la esquina superior izquierda en *fila*, *columna* para los dos primeros enteros y con *profundidad*, *anchura* para los dos enteros siguientes. La fila y la columna son relativas a la esquina superior izquierda de la ventana actual. El rectángulo se traza en el color especificado.

```
ConsoleLib.drawBoxWithColor(
 fila INT in,
 columna INT in,
 profundidad INT in,
 anchura INT in,
 color enumerationColorKind in)
```

*fila*

El número de fila en relación con la esquina superior izquierda de la ventana.

*columna*

El número de columna en relación con la esquina superior izquierda de la ventana.

*profundidad*

La profundidad o altura del recuadro.

*anchura*

La anchura del recuadro.

*color*

El color del recuadro.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## errorLine

La variable de Interfaz de usuario de consola **ConsoleLib.errorLine** controla la ubicación de la línea en la que se visualizan los mensajes de error en la pantalla **ConsoleUI**.

Tipo: INT

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## errorWindow

La variable de sistema **ConsoleLib.errorWindow** es la ventana en la que se muestra un mensaje de error de **ConsoleLib.displayError()**.

Tipo: Window

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

“displayError()” en la página 771

## errorWindowVisible

La variable de Interfaz de usuario de consola **ConsoleLib.errorWindowVisible** identifica el estado de la ventana de mensajes de error. Si el valor es **true**, la ventana es visible. Si el valor es **false**, la ventana no es visible.

Tipo: Boolean

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## formLine

La variable de sistema **ConsoleLib.formLine** es la ubicación de la línea predeterminada en la que se visualiza un formulario en la ventana. Afecta a las propiedades de las ventanas en las que se abren.

Tipo: INT

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## getKey()

La función del sistema **ConsoleLib.getKey** espera a pulsar una clave y devuelve el código de entero de la clave física pulsada. Esta función lee una clave de la entrada. El código es directo para caracteres ordinarios, el punto de código Unicode. Los resultados pueden interpretarse de forma portable comparando el resultado con el valor devuelto por **getKeyCode(String keyname)**.

**ConsoleLib.getKey( )**  
returns (*resultado* INT)

*resultado*

Un entero que representa la tecla pulsada.

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

#### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

“getKey()” en la página 775

### getKeyCode()

La función del sistema **ConsoleLib.getKeyCode** devuelve el código de entero de tecla del nombre de tecla especificado.

**ConsoleLib.getKeyCode**(*nombreTecla* STRING in)  
returns (*resultado* INT)

*resultado*

Un entero que representa el nombre de tecla.

*nombreTecla*

El nombre de la tecla lógica o física para la que se va a calcular el código de tecla correspondiente.

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

#### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

### getKeyName()

La función del sistema **ConsoleLib.getKeyName** devuelve el nombre de la tecla que representa el código de tecla de entero.

**ConsoleLib.getKeyName**(*códigoTecla* INT in)  
returns (*resultado* STRING)

*resultado*

El nombre de la tecla del código de tecla de entero.

*códigoTecla*

El código de entero de tecla.

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

#### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

### gotoField()

La función del sistema **ConsoleLib.gotoField** mueve el cursor al campo de formulario especificado. Esta función es válida en una sentencia **OpenUI** que actúa en un formulario de consola.

**ConsoleLib.gotoField**(*campoConsola1* ConsoleField in)

*campoConsola1*

El nombre de la variable de tipo ConsoleField a la que se mueve el cursor.

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

**gotoFieldByName()**

La función del sistema **ConsoleLib.gotoFieldByName** mueve el cursor al campo de formulario especificado. Esta función es válida en una sentencia **openUI** que actúa en un formulario de consola.

**ConsoleLib.gotoFieldByName**(*nombre* **STRING** *in*)

*nombre*

El nombre del campo al que se mueve el cursor.

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

**gotoMenuItem()**

La función del sistema **ConsoleLib.gotoMenuItem** mueve el cursor del menú al elemento de menú especificado. Cuando se invoca la función, se selecciona el elemento de menú especificado.

**ConsoleLib.gotoMenuItem**(*elemento* **MenuItem** *in*)

*elemento*

El elemento de menú al que se mueve el cursor.

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

**gotoMenuItemByName()**

La función del sistema **ConsoleLib.gotoMenuItemByName** mueve el cursor del menú al elemento de menú especificado. Cuando se invoca la función, se selecciona el elemento de menú especificado.

**ConsoleLib.gotoMenuItemByName**(*nombre* **STRING** *in*)

*nombre*

El nombre del elemento de menú al que se mueve el cursor.

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## hideAllMenuItems()

La función del sistema **ConsoleLib.hideAllMenuItems** oculta todos los elementos de menú del menú visualizado actualmente.

```
ConsoleLib.hideAllMenuItems()
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## hideErrorWindow()

La función del sistema **ConsoleLib.hideErrorWindow** oculta la ventana de error.

```
ConsoleLib.hideErrorWindow()
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## hideMenuItem()

La función del sistema **ConsoleLib.hideMenuItem** oculta el elemento de menú especificado de forma que el usuario no pueda seleccionarlo. Por omisión, se muestran todos los elementos de menú. El elemento oculto no se activará mediante pulsaciones de teclas.

```
ConsoleLib.hideMenuItem(elemento MenuItem in)
```

*elemento*

El elemento de menú que debe ocultarse.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## hideMenuItemByName()

La función del sistema **ConsoleLib.hideMenuItemByName** oculta el elemento de menú especificado de forma que el usuario no pueda seleccionarlo. Por omisión, se muestran todos los elementos de menú. El elemento oculto no se activará mediante pulsaciones de teclas.

```
ConsoleLib.hideMenuItemByName(nombre STRING in)
```

*nombre*

El nombre del elemento de menú a ocultar.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## interruptRequested

La variable de Interfaz de usuario de consola **ConsoleLib.interruptRequested** indica si se ha recibido o simulado una señal **INTR**. Si el valor es **true**, se ha recibido una señal **INTR**. Si el valor es **false**, no se ha recibido una señal **INTR**.

Tipo: Boolean

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## isCurrentField()

La función del sistema **ConsoleLib.isCurrentField** devuelve **yes** si el cursor está en el campo y devuelve **no** si el cursor no está en el campo. Esta función es válida en una sentencia **OpenUI** que actúa en un arrayDictionary.

```
ConsoleLib.isCurrentField(campoConsola1 ConsoleField in)
returns (resultado BOOLEAN)
```

*resultado*

**true**, si el cursor está en el campo de formulario especificado; en caso contrario, **false**.

*campoConsola1*

El nombre de la variable de tipo ConsoleField.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## isCurrentFieldByName()

La función del sistema **ConsoleLib.isCurrentFieldByName** devuelve **yes** si el cursor está en el campo; de lo contrario devuelve **no**.

Esta función es válida en una sentencia **OpenUI** que actúa en un formulario de consola.

```
ConsoleLib.isCurrentFieldByName(nombre STRING in)
returns (resultado BOOLEAN)
```

*resultado*

**true**, si el cursor está en el campo de formulario especificado; en caso contrario, **false**.

*nombre*

El valor del campo Nombre de ConsoleField.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## isFieldModified()

La función del sistema **ConsoleLib.isFieldModified** identifica para el formulario/los campos de **OpenUI** si un campo se ha modificado durante la

sentencia **OpenUI** actual. Para la matriz de pantalla de **OpenUI** (arrayDictionary), devuelve si el campo de la fila actual se ha modificado desde que el cursor entró en la fila.

Esta función solo es válida en mandatos que modifican campos y no registra el efecto de las sentencias que aparecen en una cláusula **BEFORE\_OPENUI**. Puede asignar valores a los campos en estas cláusulas sin marcar los campos como tocados.

```
ConsoleLib.isFieldModified(campoConsola1 ConsoleField in)
returns (resultado BOOLEAN)
```

*resultado*

**true**, si el cursor está en el campo de formulario especificado; en caso contrario, **false**.

*campoConsola1*

El nombre de la variable de tipo ConsoleField.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## isFieldModifiedByName()

La función del sistema **ConsoleLib.isFieldModifiedByName** identifica si el contenido de un campo nombrado se ha modificado o no.

**ConsoleLib.isFieldModifiedByName** devuelve **yes** si el usuario ha cambiado el contenido de un campo y devuelve **no** si el usuario no ha cambiado el contenido del campo.

Esta función solo es válida en mandatos que modifican campos y no registra el efecto de las sentencias que aparecen en una cláusula **BEFORE\_OPENUI**. Puede asignar valores a los campos en estas cláusulas sin marcar los campos como tocados.

```
ConsoleLib.isFieldModifiedByName(nombre STRING in)
returns (resultado BOOLEAN)
```

*resultado*

**true**, si el cursor está en el campo de formulario especificado; en caso contrario, **false**.

*nombre*

El valor del campo Nombre de ConsoleField.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## key\_accept

La variable de sistema **ConsoleLib.key\_accept** es la tecla para la finalización satisfactoria de una sentencia **OpenUI**. La tecla predeterminada es **Esc**.

Tipo: CHAR(32)

#### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

#### **key\_deleteLine**

La variable de sistema **ConsoleLib.key\_deleteLine** es la tecla para suprimir la fila actual de un arrayDictionary en un formulario de consola. La tecla predeterminada es **F2**.

Tipo: CHAR(32)

#### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

#### **key\_help**

La variable de sistema **ConsoleLib.key\_help** es la tecla para mostrar la ayuda sensible al contexto durante una sentencia **OpenUI**. La tecla predeterminada es **CRTL\_W**.

Tipo: CHAR(32)

#### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

#### **key\_insertLine**

La variable de sistema **ConsoleLib.key\_insertLine** identifica la pulsación de tecla que se utiliza para insertar una fila en un arrayDictionary de un consoleForm. La tecla predeterminada es **F1**.

Tipo: CHAR(32)

#### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

#### **key\_interrupt**

La variable de sistema **ConsoleLib.key\_interrupt** es la tecla para simular una interrupción. La tecla predeterminada es **CTRL\_C**.

Tipo: CHAR(32)

#### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

#### **key\_pageDown**

La variable de sistema **ConsoleLib.key\_pageDown** es la tecla de avance de página en un arrayDictionary de un formulario de consola. La tecla predeterminada es **F3**.

Tipo: CHAR(32)

#### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## key\_pageUp

La variable de sistema **ConsoleLib.key\_pageUp** es la tecla de retroceso de página en un arrayDictionary de un formulario de consola. La tecla predeterminada es **F4**.

Tipo: CHAR(32)

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## key\_quit

La variable de sistema **ConsoleLib.key\_quit** es la tecla para salir del programa sin validar la entrada de usuario. La tecla predeterminada es **CTRL\_\**\****.

Tipo: CHAR(32)

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## lastKeyTyped()

La función del sistema **ConsoleLib.lastKeyTyped** devuelve el código de entero de la última tecla física que se ha pulsado en el teclado.

```
ConsoleLib.lastKeyTyped()
returns (resultado INT)
```

*resultado*

Un entero que representa la última tecla pulsada.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## menuLine

La variable de sistema **ConsoleLib.menuLine** contiene la ubicación de línea en la que los menús se visualizan en una ventana. Afecta a las propiedades de las ventanas en las que se abren.

Tipo: INT

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## messageLine

La variable de sistema **ConsoleLib.messageLine** es la ubicación de ventana en la que se visualizan los mensajes.

Tipo: INT

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## messageResource

La variable de sistema **ConsoleLib.messageResource** es el nombre de archivo del paquete compuesto de recursos del que se cargan la ayuda y otros mensajes. Si esta variable no tiene ningún valor, el entorno de ejecución EGL inspecciona el archivo identificado en la propiedad de tiempo de ejecución Java **vgj.messages.file**.

Tipo: CHAR(255)

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“Interfaz de usuario de consola” en la página 177

### Consulta relacionada

“Componentes de ConsoleUI y variables relacionadas” en la página 180

“Biblioteca ConsoleLib de EGL” en la página 757

“Propiedades de ejecución de Java (detalles)” en la página 540

## nextField()

La función del sistema **ConsoleLib.nextField** mueve el cursor al campo de formulario siguiente según el orden de desplazamiento de campos definido. Esta función es válida en una sentencia **openUI** que actúa en un formulario de consola.

```
ConsoleLib.nextField()
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## openWindow()

La función del sistema **ConsoleLib.openWindow** hace visible una ventana y la añade a la parte superior de la pila de ventanas.

```
ConsoleLib.openWindow(ventana1 Window inOut)
```

*ventana1*

Una variable de tipo Window.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## openWindowByName()

La función del sistema **ConsoleLib.openWindowByName** hace visible una ventana y la añade a la parte superior de la pila de ventanas.

```
ConsoleLib.openWindowByName(nombre STRING in)
```

*nombre*

El valor del campo Nombre de ventana.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## openWindowWithForm()

La función del sistema **ConsoleLib.openWindowWithForm** hace visible una ventana, la añade a la parte superior de la pila de ventanas y visualiza el formulario en la ventana. La ventana se redimensiona para ajustarse al formulario.

```
ConsoleLib.openWindowWithForm(
 ventana1 Window inOut,
 formulario ConsoleForm in)
```

*ventana1*

Una variable de tipo Window.

*formulario*

Una variable de tipo ConsoleForm.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## openWindowWithFormByName()

La función del sistema **ConsoleLib.openWindowWithFormByName** activa una ventana, la hace visible y visualiza el formulario de consola especificado. La ventana se redimensiona para ajustarse al formulario.

```
ConsoleLib.openWindowWithFormByName(
 nombreVentana STRING in,
 nombreFormulario STRING in)
```

*nombreVentana*

El valor del campo Nombre de ventana.

*nombreFormulario*

El valor del campo Nombre de ConsoleForm.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## previousField()

La función del sistema **ConsoleLib.previousField** mueve el cursor al campo de formulario anterior según el orden de tabulación de campos definido.

```
ConsoleLib.previousField()
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## **promptLine**

La variable de sistema **ConsoleLib.promptLine** es la línea predeterminada en la que se visualizan las solicitudes en una ventana. Esto afecta a las propiedades de las ventanas en las que se abren.

Tipo: INT

### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## **promptLineMode()**

La función de sistema **ConsoleLib.promptLineMode** visualiza la serie en modalidad de línea y espera la entrada de usuario, que se envía cuando el usuario pulsa **Intro**.

```
ConsoleLib.promptLineMode(mensaje String in)
returns (resultado STRING)
```

*resultado*

La entrada del usuario.

*mensaje*

La frase a visualizar.

### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## **quitRequested**

La variable de sistema **ConsoleLib.quitRequested** indica que se ha recibido (o simulado) una señal **QUIT**. Si el valor es **true**, se ha recibido una señal **QUIT**. Si el valor es **false**, no se ha recibido una señal **QUIT**.

Tipo: Boolean

### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## **screen**

La variable de sistema **ConsoleLib.screen** define automáticamente una ventana predeterminada sin bordes. Las dimensiones de la pantalla son iguales a las dimensiones de la superficie de visualización disponible.

Tipo: Window

### **Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

## **scrollDownLines()**

La función de sistema **ConsoleLib.scrollDownLines** desplaza los datos en pantalla hacia el final de los datos.

```
ConsoleLib.scrollDownLines(númeroLíneas INT in)
```

*númeroLíneas*

El número de líneas a desplazar hacia abajo.

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

**scrollDownPage()**

La función de sistema **ConsoleLib.scrollDownPage** desplaza los datos en pantalla una página hacia el final de los datos.

**ConsoleLib.scrollDownPage( )**

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

**scrollUpLines()**

La función de sistema **ConsoleLib.scrollUpLines** desplaza los datos en pantalla hacia el principio de los datos.

**ConsoleLib.scrollUpLines(*númeroLíneas* INT in)**

*númeroLíneas*

El número de líneas a desplazar hacia arriba.

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

**scrollUpPage()**

La función de sistema **ConsoleLib.scrollUpPage** desplaza los datos en pantalla una página hacia el principio de los datos.

**ConsoleLib.scrollUpPage( )**

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca ConsoleLib de EGL” en la página 757

**setArrayLine()**

La función del sistema **ConsoleLib.setArrayLine** mueve la selección al registro de programa especificado. Si es necesario, los datos se desplazan para hacer visible el registro seleccionado.

**ConsoleLib.setArrayLine(*númeroRegistro* INT in)**

*númeroRegistro*

El registro a seleccionar.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## setCurrentArrayCount()

La función de sistema **ConsoleLib.setCurrentArrayCount** especifica el número de filas que existe en una matriz dinámica que está enlazada a un `arrayDictionary` en pantalla. Esta función sólo es útil si la invoca antes de emitir la sentencia **openUI** que utiliza el `arrayDictionary`.

```
ConsoleLib.setCurrentArrayCount(cuenta INT in)
```

*cuenta*

El número de entradas de matriz cuando empieza la sentencia `openUI`.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

“`openUI`” en la página 620

## showAllMenuItems()

La función del sistema **ConsoleLib.showAllMenuItems** muestra todos los elementos de menú del menú visualizado actualmente.

```
ConsoleLib.showAllMenuItems()
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## showHelp()

La función del sistema **ConsoleLib.showHelp** muestra un mensaje de ayuda. El argumento de serie es la clave del mensaje en el paquete compuesto de recursos configurado con el campo **ConsoleLib.messageResource**.

```
ConsoleLib.showHelp(teclaAyuda STRING in)
```

*teclaAyuda*

La tecla que busca un mensaje de ayuda en el texto.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## showMenuItem()

La función del sistema **ConsoleLib.showMenuItem** muestra el elemento de menú especificado para que pueda seleccionarlo el usuario. Por omisión, se muestran todos los elementos de menú.

**ConsoleLib.showMenuItem**(*elemento MenuItem in*)

*elemento*

El elemento de menú a mostrar.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## showMenuItemByName

La función del sistema **ConsoleLib.showMenuItemByName** muestra el elemento de menú especificado para que pueda seleccionarlo el usuario. Por omisión, se muestran todos los elementos de menú.

**ConsoleLib.showMenuItemByName**(*nombre STRING in*)

*nombre*

El valor del campo Nombre de MenuItem.

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## sqlInterrupt

Para la variable de sistema **ConsoleLib.sqlInterrupt**, si el valor es **yes**, el usuario puede interrumpir las sentencias SQL que se están procesando. Si el valor es **no**, el usuario solo puede interrumpir las sentencias **OpenUI**. La variable **sqlInterrupt** se utiliza en combinación con las variables *deferInterrupt* y *deferQuit*.

Tipo: Boolean

### Consulta relacionada

“Biblioteca ConsoleLib de EGL” en la página 757

## Biblioteca ConverseLib de EGL

La biblioteca Converse proporciona las funciones que se muestran en la tabla siguiente.

Función	Descripción
clearScreen ()	Borra la pantalla, ya que es útil antes de que el programa emita una sentencia converse en una aplicación de texto.
displayMsgNum ( <i>númeroMsj</i> )	Recupera un valor de la tabla de mensajes del programa. El mensaje se presentará la próxima vez que se presente un formulario mediante una sentencia <b>converse</b> , <b>display</b> , <b>print</b> o <b>show</b> .
<i>result</i> = fieldInputLength ( <i>textField</i> )	Devuelve el número de caracteres que el usuario escribió en el campo de entrada cuando el formulario de texto se presentó por última vez. Dicho número no incluye los blancos o nulos iniciales o finales.

Función	Descripción
pageEject ()	Avanza la salida del formulario de impresión hasta el principio de la página siguiente, ya que es útil antes de que el programa emita una sentencia print.
validationFailed ( <i>númeroMsj</i> )	<ul style="list-style-type: none"> <li>• Si se invoca en una función de validación de campos de una aplicación de texto, <b>ConverseLib.validationFailed</b> provoca la re-presentación del formulario de texto recibido una vez procesadas todas las funciones de validación. La última función <b>ConverseLib.validationFailed</b> invocada determina el mensaje que se visualiza.</li> <li>• Si se invoca fuera de una función de validación, <b>ConverseLib.validationFailed</b> presentará el mensaje especificado la próxima vez que se presente un formulario mediante una sentencia <b>converse</b>, <b>display</b>, <b>print</b> o <b>show</b>. En este caso, el comportamiento es como el de <b>ConverseLib.displayMsgNum</b>.</li> </ul>

## clearScreen()

La función de sistema **ConverseLib.clearScreen** borra la pantalla, lo cual resulta de utilidad antes de que el programa emita una sentencia converse en una aplicación de texto.

**ConverseLib.clearScreen( )**

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“converse” en la página 570

“Biblioteca ConverseLib de EGL” en la página 788

## displayMsgNum()

La función de sistema **ConverseLib.displayMsgNum** recupera un valor de la tabla de mensajes del programa. El mensaje se presentará la próxima vez que se presente un formulario mediante una sentencia **converse**, **display**, **print** o **show**.

Si es posible, la presentación del mensaje se realiza en el propio formulario, en el campo al que hace referencia la propiedad de formulario **msgField**. Si la propiedad de formulario **msgField** no indica ningún valor, el mensaje se visualiza antes de la visualización del formulario, en una pantalla modal independiente o en una página imprimible.

**ConverseLib.displayMsgNum** toma como único argumento un valor que se compara con cada casilla de la primera columna de la *tabla de mensajes* del programa, que es la tabla de datos a la que hace referencia la propiedad **msgTablePrefix** del programa. El mensaje recuperado por esa función se encuentra en la segunda columna de la misma fila.

**ConverseLib.displayMsgNum(*númeroMsj* INT in)**

*númeroMsj*

El mensaje se recupera de la tabla de mensajes según el número. El argumento debe ser un literal entero o un elemento de tipo primitivo SMALLINT o INT o el equivalente BIN.

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Biblioteca ConverseLib de EGL” en la página 788

### **fieldInputLength()**

La función de sistema **ConverseLib.fieldInputLength** devuelve el número de caracteres que el usuario ha tecleado en el campo de entrada cuando el formulario de texto se ha presentado por última vez. Dicho número no incluye los blancos o nulos iniciales o finales.

Si el campo está en su estado definido originariamente, la función devuelve una longitud de 0. Por ejemplo, si el campo contiene la propiedad *value* y no se ha modificado de ninguna manera durante la ejecución, se calculará la longitud como 0. La sentencia *set form initial* restablece el campo a su estado definido originariamente. Si el campo no está en su estado definido originariamente, la longitud se calcula basándose en lo visualizado o entrado en la última sentencia de inversión.

```
ConverseLib.fieldInputLength(campoTexto TestFormField in)
returns (resultado INT)
```

*resultado*

El número de caracteres que ha tecleado el usuario.

*campoTexto*

El nombre del campo de texto.

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Biblioteca ConverseLib de EGL” en la página 788

### **pageEject()**

La función de sistema **ConverseLib.pageEject** avanza la salida del formulario de impresión hacia el principio de la página siguiente, lo cual resulta de utilidad antes de que el programa emita una sentencia print.

```
ConverseLib.pageEject()
```

Para obtener otros detalles relativos a la impresión, consulte el apartado

*Formularios de impresión.*

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

“Formularios de impresión” en la página 156

#### **Consulta relacionada**

“Biblioteca ConverseLib de EGL” en la página 788

“print” en la página 632

## validationFailed()

La función de sistema **ConverseLib.validationFailed** puede utilizarse de dos formas:

- Si se invoca en una función de validación de campos de una aplicación de texto, **ConverseLib.validationFailed** provoca la re-presentación del formulario de texto recibido una vez procesadas todas las funciones de validación. La última función **ConverseLib.validationFailed** invocada determina el mensaje que se visualiza. Si es posible, la presentación del mensaje se realiza en el propio formulario, en el campo al que hace referencia la propiedad de formulario **msgField**. Si la propiedad de formulario **msgField** no indica ningún valor, el mensaje se visualiza antes de la visualización del formulario, en una pantalla modal independiente.
- Si se invoca fuera de una función de validación, **ConverseLib.validationFailed** presentará el mensaje especificado la próxima vez que se presente un formulario mediante una sentencia **converse**, **display**, **print** o **show**. En este caso, el comportamiento es como el de **ConverseLib.displayMsgNum**.

En cualquier caso, el valor asignado a **ConverseLib.validationFailed** se almacena en la variable de sistema **ConverseVar.validationMsgNum**.

**ConverseLib.validationFailed**([*númeroMsj* INT *in*])

*númeroMsj*

El número del mensaje que debe visualizarse. El argumento debe ser un literal entero o un elemento de tipo primitivo SMALLINT o INT o el equivalente BIN. Este número se compara con cada casilla de la primera columna de la *tabla de mensajes* del programa, que es la tabla de datos a la que hace referencia la propiedad **msgTablePrefix** del programa. El mensaje recuperado se encuentra en la segunda columna de la misma fila.

El número del mensaje es 9999 por omisión.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“displayMsgNum()” en la página 789

“validationMsgNum” en la página 926

“Biblioteca ConverseLib de EGL” en la página 788

## Biblioteca DateTimeLib de EGL

Las variables de sistema de fecha y hora permiten recuperar la fecha y hora del sistema en diversos formatos, como se muestra en la tabla siguiente.

Variable de sistema	Descripción
<i>result</i> = currentDate ()	Contiene la fecha del sistema actual en formato gregoriano de ocho dígitos (aaaaMMdd); puede asignar esta variable del sistema a una variable de tipo DATE.
<i>result</i> = currentTime ()	Contiene la hora del sistema actual en formato juliano de seis dígitos (HHmmss); puede asignar esta variable del sistema a un tipo TIME.

Variable de sistema	Descripción
<i>result</i> = <i>currentTimeStamp</i> ()	Contiene la hora y la fecha del sistema actual como una indicación de la hora en formato juliano de veinte dígitos (aaaaMMddHHmmssffffff); puede asignar esta variable del sistema a una variable de tipo <code>TIMESTAMP</code> .
<i>result</i> = <i>dateOf</i> ( <i>aTimeStamp</i> )	Devuelve una fecha que se deriva de una variable de tipo <code>TIMESTAMP</code> .
<i>result</i> = <i>dateValue</i> ( <i>dateAsString</i> )	Devuelve un valor <code>DATE</code> que corresponde a una serie de entrada.
<i>result</i> = <i>dateValueFromGregorian</i> ( <i>gregorianIntegerDate</i> )	Devuelve un valor <code>DATE</code> que corresponde a una representación de entero de una fecha gregoriana.
<i>result</i> = <i>dateValueFromJulian</i> ( <i>julianIntegerDate</i> )	Devuelve un valor <code>DATE</code> que corresponde a una representación de entero de una fecha juliana.
<i>result</i> = <i>dayOf</i> ( <i>aTimeStamp</i> )	Devuelve un entero positivo que representa un día del mes, derivado de una variable de tipo <code>TIMESTAMP</code> .
<i>result</i> = <i>extend</i> ( <i>extensionField</i> [, <i>mask</i> ])	Convierte una indicación de la hora, una hora o una fecha en un valor de indicación de la hora más largo o más corto.
<i>result</i> = <i>intervalValue</i> ( <i>intervalAsString</i> )	Devuelve un valor <code>INTERVAL</code> que refleja un literal o una constante de serie.
<i>result</i> = <i>intervalValueWithPattern</i> ( <i>intervalAsString</i> [, <i>intervalMask</i> ])	Devuelve un valor <code>INTERVAL</code> que refleja un literal o una constante de serie y que se construye basándose en una máscara de intervalo que usted especifique.
<i>result</i> = <i>mdy</i> ( <i>month</i> , <i>day</i> , <i>year</i> )	Devuelve un valor <code>DATE</code> derivado de tres enteros que representan el mes, el día del mes y el año de una fecha de calendario.
<i>result</i> = <i>monthOf</i> ( <i>aTimeStamp</i> )	Devuelve un entero positivo que representa un mes, derivado de una variable de tipo <code>TIMESTAMP</code> .
<i>result</i> = <i>timeOf</i> ([ <i>aTimeStamp</i> ])	Devuelve una serie que representa la hora del día, que se deriva de una variable <code>TIMESTAMP</code> o del reloj del sistema.
<i>result</i> = <i>timeStampFrom</i> ( <i>tsDate</i> <i>tsTime</i> )	Contiene la hora y la fecha del sistema actual como una indicación de la hora en formato juliano de veinte dígitos (aaaaMMddHHmmssffffff); puede asignar esta variable del sistema a una variable de tipo <code>TIMESTAMP</code> .
<i>result</i> = <i>timeStampValue</i> ( <i>timeStampAsString</i> )	Devuelve un valor <code>TIMESTAMP</code> que refleja un literal o una constante de serie.
<i>result</i> = <i>timeStampValueWithPattern</i> ( <i>timeStampAsString</i> [, <i>timeStampMask</i> ])	Devuelve un valor <code>TIMESTAMP</code> que refleja una serie y se construye basándose en una máscara de indicación de la hora que usted especifique.
<i>result</i> = <i>timeValue</i> ( <i>timeAsString</i> )	Devuelve un valor <code>TIME</code> que refleja un literal o una constante de serie.

Variable de sistema	Descripción
<i>result = weekdayOf (aTimeStamp)</i>	Devuelve un entero positivo (0-6) que representa un día de semana, derivado de una variable de tipo <code>TIMESTAMP</code> .
<i>result = yearOf (aTimeStamp)</i>	Devuelve un entero que representa un año, derivado de una variable de tipo <code>TIMESTAMP</code> .

Para establecer una variable de fecha, hora o indicación de la hora, puede asignar `VGVar.currentGregorianDate`, `DateTimeLib.currentTime` y `DateTimeLib.currentTimeStamp`, respectivamente. Las funciones que devuelven el texto de caracteres con formato no pueden utilizarse para este propósito.

#### Consulta relacionada

“Sentencias EGL” en la página 88

#### **currentDate()**

La función del sistema `DateTimeLib.currentDate` lee el reloj del sistema y devuelve un valor `DATE` que representa la fecha del calendario actual. La función sólo devuelve la fecha actual, no la hora del día.

```
DateTimeLib.currentDate()
returns (resultado DATE)
```

*resultado*

Un valor `DATE` que representa la fecha del calendario actual.

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

#### Consulta relacionada

“DATE” en la página 41

“Biblioteca `DateTimeLib` de EGL” en la página 791

#### **currentTime()**

La función del sistema `DateTimeLib.currentTime` recupera la hora del sistema actual en formato juliano de seis dígitos (HHmmss). El valor se actualiza automáticamente cada vez que el programa hace referencia al mismo.

```
DateTimeLib.currentTime()
returns (resultado TIME)
```

*resultado*

Un valor `TIME` que representa la hora actual del sistema.

Puede utilizar `DateTimeLib.currentTime` de las siguientes maneras:

- Como origen de una sentencia `assignment` o `move`
- Como argumento de una sentencia `return`

Las características de `DateTimeLib.currentTime` son las siguientes:

#### Tipo primitivo

`TIME`

#### Longitud de datos

6

#### ¿Se guarda el valor a lo largo de los segmentos?

No

**Ejemplo:**

```
myTime = DateTimeLib.currentTime;
```

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca DateTimeLib de EGL” en la página 791

**currentTimeStamp()**

La función del sistema **DateTimeLib.currentTimeStamp** recupera la hora y la fecha del sistema actual como una indicación de la hora en formato de veinte dígitos (aaaaMMddHHmmssffffff). El valor se actualiza automáticamente cada vez que el programa hace referencia al mismo.

```
DateTimeLib.currentTimeStamp()
returns (resultado TIMESTAMP)
```

*resultado*

Un valor **TIMESTAMP** que representa la hora y fecha actuales del sistema.

Puede utilizar **DateTimeLib.currentTimeStamp** de las siguientes maneras:

- Como origen de una sentencia assignment o **move**
- Como argumento de una sentencia **return**

Las características de **DateTimeLib.currentTimeStamp** son las siguientes:

**Tipo primitivo**

**TIMESTAMP**

**Longitud de datos**

20

**¿Se guarda el valor a lo largo de los segmentos?**

No

**Ejemplo:**

```
myTimeStamp = DateTimeLib.currentTimeStamp;
```

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca DateTimeLib de EGL” en la página 791

**dateOf()**

La función de sistema **DateTimeLib.dateOf** devuelve un valor **DATE** derivado de una variable de tipo **TIMESTAMP**.

```
DateTimeLib.dateOf(aTimeStamp TIMESTAMP in)
returns (resultado DATE)
```

*resultado*

Un valor **DATE**.

*aTimeStamp*

El valor del que se deriva la fecha.

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“DATE” en la página 41

“Biblioteca DateTimeLib de EGL” en la página 791

### dateValue()

La función **DateTimeLib.dateValue** devuelve un valor DATE que corresponde a una serie.

```
DateTimeLib.dateValue(fechaComoSerie STRING in)
returns (resultado DATE)
```

*resultado*

Una variable de tipo DATE.

*fechaComoSerie*

Un literal o una constante de serie que contiene dígitos que reflejan la máscara "aaaaMMdd". Para conocer más detalles consulte el apartado *DATE*.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“DATE” en la página 41

“Expresiones de fecha y hora” en la página 495

### dateValueFromGregorian()

La función **DateTimeLib.dateValueFromGregorian** devuelve un valor DATE que corresponde a una representación de entero de una fecha gregoriana.

```
DateTimeLib.dateValueFromGregorian(
 fechaEnteroGregoriana INT in)
returns (resultado DATE)
```

*resultado*

Una variable de tipo DATE.

*fechaEnteroGregoriana*

Un valor numérico de VisualAge Generator que representa una fecha gregoriana en el formato 00AAAMMDD u 00AAMMDD.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“DATE” en la página 41

“Biblioteca DateTimeLib de EGL” en la página 791

### dateValueFromJulian()

La función **DateTimeLib.dateValueFromJulian** devuelve un valor DATE que corresponde a una representación de entero de una fecha juliana.

```
DateTimeLib.dateValueFromJulian(
 fechaEnteroJuliana INT in)
returns (resultado DATE)
```

*resultado*

Una variable de tipo DATE.

*fechaEnteroJuliana*

Un valor numérico de VisualAge Generator que representa una fecha juliana en el formato 00AAAADDD u 00AADD.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“DATE” en la página 41

“Biblioteca DateTimeLib de EGL” en la página 791

### dayOf()

La función de sistema **DateTimeLib.dayOf** devuelve un entero positivo que representa un día (1-7), derivado de una variable de tipo **TIMESTAMP**.

```
DateTimeLib.dayOf(aTimeStamp TIMESTAMP in)
returns (resultado INT)
```

*resultado*

Un entero positivo que corresponde al día del mes.

*aTimeStamp*

La variable de la que se deriva el día.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“DATE” en la página 41

“Biblioteca DateTimeLib de EGL” en la página 791

### extend()

La función del sistema **DateTimeLib.extend** convierte una indicación de la hora, una hora, o una fecha en un valor de indicación de la hora más largo o más corto. Ejemplos:

- Si tiene una indicación de la hora de entrada definida como “ddHH” (día y hora) y proporciona una máscara de indicación de la hora “ddHHmm” (día, hora y minuto), **DateTimeLib.extend** devuelve un valor ampliado que coincide con la máscara
- Si tiene una indicación de la hora de entrada definida como “aaaaMMddHHmmss” (año, mes, día, hora, minuto y segundo) y proporciona una máscara de indicación de la hora “aaa” (año), **DateTimeLib.extend** devuelve un valor abreviado que coincide con la máscara

```
DateTimeLib.extend(
 campoAmpliación dateOrTimeOrTimeStamp in
 [, máscara outputTimeStampMask in
])
returns (resultado TIMESTAMP)
```

*resultado*

Una variable de tipo **TIMESTAMP**.

*campoAmpliación*

El nombre de un campo de tipo **TIMESTAMP**, **TIME** o **DATE**. El campo contiene el valor que debe ampliarse o abreviarse.

*máscara*

Un literal o una constante de serie que define la máscara del valor de indicación de la hora devuelto por la función. Para conocer más detalles, consulte el apartado **TIMESTAMP**.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Expresiones de fecha y hora” en la página 495

### intervalValue()

La función `DateTimeLib.intervalValue` devuelve un valor de `INTERVAL` que refleja un literal o una constante de serie y que se construye basándose en la máscara de intervalo por omisión, que es `aaaaMM`.

La serie de entrada debe contener seis dígitos. Los cuatro primeros dígitos representan el número de años del intervalo, y los dos últimos representan el número de meses.

Si desea especificar una máscara que no sea `aaaaMM`, invoque

### `DateTimeLib.intervalValueWithPattern`.

```
DateTimeLib.intervalValue(intervaloComoSerie STRING in)
returns (resultado INTERVAL)
```

*resultado*

Una variable de tipo `INTERVAL`

*intervaloComoSerie*

Un literal o una constante de serie que contiene seis dígitos cuyo significado viene indicado por la máscara de intervalo, `aaaaMM`

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Expresiones de fecha y hora” en la página 495

“`INTERVAL`” en la página 42

“`intervalValueWithPattern()`”

### intervalValueWithPattern()

La función del valor de fecha y hora `DateTimeLib.intervalValueWithPattern` devuelve un valor `INTERVAL` que refleja un literal o una constante de serie y que (opcionalmente) se construye basándose en una máscara de intervalo que especifique. Si la máscara es `aaaa`, por ejemplo, la serie de entrada debe contener cuatro dígitos y esos cuatro dígitos representan el número de años representados en el intervalo.

```
DateTimeLib.intervalValueWithPattern(
 intervaloComoSerie STRING in
 [, máscaraIntervalo STRING in
])
returns (resultado INTERVAL)
```

*resultado*

Una variable de tipo `INTERVAL`.

*intervaloComoSerie*

Un literal o una constante de serie que contiene dígitos cuyo significado viene indicado por la máscara de intervalo.

*máscaraIntervalo*

Especifica una máscara de intervalo que da significado a cada dígito del primer parámetro. La máscara por omisión es `aaaaMM`. Para obtener otros detalles, consulte el apartado `INTERVAL`.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Expresiones de fecha y hora” en la página 495

“INTERVAL” en la página 42

### mdy()

El operador mdy devuelve un valor DATE derivado de tres enteros que representan el mes, el día del mes y el año de una fecha de calendario.

```
DateTimeLib.mdy(
 mes INT in,
 día INT in,
 año INT in)
returns (resultado DATE)
```

*resultado*

Un valor DATE.

*mes*

Un entero en el rango 1 a 12, que representa el mes.

*día*

Un entero que representa el día del mes, en un rango de 1 a 28, 29, 30 ó 31, dependiendo del mes.

*año*

Un entero de cuatro dígitos que representa el año.

Se produce un error si especifica valores fuera del rango de días y meses en el calendario o si el número de operandos no es tres. Debe incluir los tres operandos de expresión enteros entre paréntesis y separados por comas, tal como haría si MDY( ) fuera una función. La tercera expresión no puede ser una abreviatura del año. Por ejemplo, 99 especifica un año del primer siglo, hace aproximadamente 1.900 años.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“DATE” en la página 41

“Biblioteca DateTimeLib de EGL” en la página 791

### monthOf()

La función de sistema **DateTimeLib.monthOf** devuelve un entero positivo que representa un mes, derivado de una variable de tipo TIMESTAMP.

```
DateTimeLib.monthOf(aTimeStamp TIMESTAMP in)
returns (resultado INT)
```

*resultado*

Un entero positivo que representa un mes.

*aTimeStamp*

La variable TIMESTAMP de la que se deriva el mes.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“DATE” en la página 41

“Biblioteca DateTimeLib de EGL” en la página 791

## timeOf()

La función de sistema **DateTimeLib.timeOf** devuelve una serie que representa la hora del día, que se deriva de una variable **TIMESTAMP** o del reloj del sistema.

```
DateTimeLib.timeOf([aTimeStamp TIMESTAMP in])
returns (resultado STRING)
```

### *resultado*

La parte de hora del día del argumento *aTimeStamp*, de acuerdo con un reloj de 24 horas y el siguiente formato:

hh:mm:ss

*hh* La hora como serie de dos dígitos.

*mm*

El minuto como serie de dos dígitos.

*ss* El segundo como serie de dos dígitos.

### *aTimeStamp*

Un valor **DATETIME**. Si no se especifica ningún valor, el operador devuelve una serie de caracteres que representan la hora actual.

## Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

## Consulta relacionada

“DATE” en la página 41

“Biblioteca **DateTimeLib** de EGL” en la página 791

## timeStampFrom()

La función **DateTimeLib.timeStampFrom** devuelve un valor **TIMESTAMP** que se construye a partir de un valor **DATE** y **TIME** que especifique.

```
DateTimeLib.timeStampFrom(
 tsDate DATE in,
 tsTime TIME in)
returns (resultado TIMESTAMP)
```

### *resultado*

Un valor de tipo **TIMESTAMP**.

### *tsDate*

Una variable de tipo **DATE**.

### *tsTime*

Una variable de tipo **TIME**.

## Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

## Consulta relacionada

“Expresiones de fecha y hora” en la página 495

“Biblioteca **DateTimeLib** de EGL” en la página 791

“**TIMESTAMP**” en la página 44

## timeStampValue()

La función **DateTimeLib.timeStampValue** devuelve un valor de **TIMESTAMP** que refleja un literal o una constante de serie y que se construye basándose en la máscara de indicación de la hora por omisión, que es *aaaaMMddHHmmss*.

La serie de entrada debe contener catorce dígitos:

- Los cuatro primeros dígitos representan el año
- Los dos siguientes representan el mes numérico
- Los dos siguientes representan el día del mes
- Los dos siguientes representan el número de horas (de 00 a 24)
- Los dos siguientes representan el número de minutos dentro de la hora
- Los dos últimos representan el número de segundos dentro del minuto

Si desea especificar una máscara que no sea *aaaaMMddHHmmss*, invoque **DateTimeLib.timestampValueWithPattern**.

```
DateTimeLib.timestampValue(indicaciónHoraComoSerie STRING in)
returns (resultado TIMESTAMP)
```

*resultado*

Una variable de tipo **TIMESTAMP**.

*indicaciónHoraComoSerie*

Un literal o una constante de serie que contiene catorce dígitos cuyo significado viene indicado por la máscara de indicación de la hora, *aaaaMMddHHmmss*.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Expresiones de fecha y hora” en la página 495

“TIMESTAMP” en la página 44

“timeStampValueWithPattern()”

### timeStampValueWithPattern()

La función **DateTimeLib.timeStampValueWithPattern** devuelve un valor de **TIMESTAMP** que refleja un literal o una constante de serie y que (opcionalmente) se construye basándose en una máscara de indicación de la hora que especifique. Si la máscara es “aaaa”, por ejemplo, la serie de entrada debe contener cuatro dígitos y los dígitos que representan el valor del año en la indicación de la hora.

```
DateTimeLib.timeStampValueWithPattern(
 indicaciónHoraComoSerie STRING in
 [, máscaraIndicaciónHora STRING in
])
returns (resultado TIMESTAMP)
```

*resultado*

Una variable de tipo **TIMESTAMP**.

*indicaciónHoraComoSerie*

Un literal o una constante de serie que contiene dígitos cuyo significado viene indicado por la máscara de indicación de la hora.

*máscaraIndicaciónHora*

Especifica una máscara de indicación de la hora que da significado a cada dígito del primer parámetro. La máscara por omisión es *aaaaMMddHHmmss*. Para obtener otros detalles, consulte el apartado **TIMESTAMP**.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Expresiones de fecha y hora” en la página 495

“TIMESTAMP” en la página 44

## **timeValue()**

La función de valor de fecha y hora **DateTimeLib.timeValue** devuelve un valor TIME que refleja un literal o una constante de serie.

```
DateTimeLib.timeValue(horaComoSerie STRING in)
returns (resultado TIME)
```

*resultado*

Una variable de tipo TIME.

*horaComoSerie*

Un literal o una constante de serie que contiene dígitos que reflejan la máscara "HHmmss". Para obtener detalles, consulte el apartado *TIME*.

### **Conceptos relacionados**

"Diagrama de sintaxis para funciones EGL" en la página 754

### **Consulta relacionada**

"Expresiones de fecha y hora" en la página 495

"TIME" en la página 43

## **weekdayOf()**

La función de sistema **DateTimeLib.weekdayOf** devuelve un entero positivo que representa un día de la semana, derivado de una variable de tipo **TIMESTAMP**. El número 0 representa domingo, 1 representa lunes, etc.

```
DateTimeLib.weekdayOf(aTimeStamp TIMESTAMP in)
returns (resultado INT)
```

*resultado*

Un entero positivo de 0 a 6.

*aTimeStamp*

La variable **TIMESTAMP** de la que se deriva el día.

### **Conceptos relacionados**

"Diagrama de sintaxis para funciones EGL" en la página 754

### **Consulta relacionada**

"DATE" en la página 41

"Biblioteca DateTimeLib de EGL" en la página 791

## **yearOf()**

La función de sistema **DateTimeLib.yearOf** devuelve un entero de cuatro dígitos que representa el año, derivado de una variable de tipo **TIMESTAMP**.

```
DateTimeLib.yearOf(aTimeStamp TIMESTAMP in)
returns (resultado INT)
```

*resultado*

El entero que representa el año.

*aTimeStamp*

La variable **TIMESTAMP** de la que se deriva el año.

### **Conceptos relacionados**

"Diagrama de sintaxis para funciones EGL" en la página 754

### **Consulta relacionada**

"DATE" en la página 41

"Biblioteca DateTimeLib de EGL" en la página 791

## Biblioteca J2EELib de EGL

La tabla siguiente lista las funciones del sistema de la biblioteca J2EELib.

Función	Descripción
<code>clearRequestAttr</code> ( <i>clave</i> )	Elimina el argumento asociado a la clave especificada en el objeto de petición.
<code>clearSessionAttr</code> ( <i>clave</i> )	Elimina el argumento asociado a la clave especificada en el objeto de sesión.
<code>getRequestAttr</code> ( <i>clave, argumento</i> )	Utiliza una clave especificada para recuperar un argumento del objeto de petición en una variable especificada.
<code>getSessionAttr</code> ( <i>clave, argumento</i> )	Utiliza una clave especificada para recuperar un argumento del objeto de sesión en una variable especificada.
<code>setRequestAttr</code> ( <i>clave, argumento</i> )	Utiliza una clave especificada para colocar un argumento especificado en el objeto de petición.
<code>setSessionAttr</code> ( <i>clave, argumento</i> )	Utiliza una clave especificada para colocar un argumento especificado en el objeto de sesión.

### **clearRequestAttr()**

La función de sistema **J2EELib.clearRequestAttr** elimina el argumento que está asociado con la clave especificada del objeto de petición. Esta función se utiliza en los PageHandlers y en los programas que se ejecutan en aplicaciones Web.

Puede colocar establecer un argumento en el objeto de petición mediante la función de sistema **J2EELib.setRequestAttr**. Puede recuperar el argumento mediante la función de sistema **J2EELib.getRequestAttr**.

**J2EELib.clearRequestAttr**(*clave* **STRING** *in*)

*clave*

Un literal de serie o una expresión de tipo String

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

“PageHandler” en la página 194

#### **Consulta relacionada**

“Biblioteca J2EELib de EGL”

“getRequestAttr()” en la página 803

“setRequestAttr()” en la página 804

### **clearSessionAttr()**

La función de sistema **J2EELib.clearSessionAttr** elimina el argumento que está asociado con la clave especificada del objeto de sesión. Esta función se utiliza en los PageHandlers y en los programas que se ejecutan en aplicaciones Web.

Puede colocar un argumento en el objeto de sesión mediante la función de sistema **J2EELib.setSessionAttr**. Puede recuperar el argumento mediante la función de sistema **J2EELib.getSessionAttr**.

**J2EELib.clearSessionAttr**(*clave* **STRING** *in*)

*clave*

Un literal de serie o una expresión de tipo STRING

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

“PageHandler” en la página 194

#### **Consulta relacionada**

“Biblioteca J2EELib de EGL” en la página 802

“getSessionAttr()”

“setSessionAttr()” en la página 804

#### **getRequestAttr()**

La función de sistema **J2EELib.getRequestAttr** utiliza una clave especificada para recuperar un argumento del objeto de petición en una variable especificada. Esta función se utiliza en los PageHandlers y en los programas que se ejecutan en aplicaciones Web.

Si no se encuentra un objeto con la clave especificada, la variable destino no sufre cambios. Si el objeto recuperado es de un tipo incorrecto, se lanza una excepción y el programa o el PageHandler finalizan.

Puede colocar un argumento en el objeto de petición mediante la función de sistema **J2EELib.setRequestAttr**. El objeto de argumento colocado en la colección de peticiones del servlet está disponible para su acceso mientras la petición del servlet sea válida. Someter un formulario desde una página provoca la creación de una nueva petición.

```
J2EELib.getRequestAttr(
 clave STRING in,
 argumento attribute inOut)
```

*clave*

Un literal de carácter o un elemento de cualquier tipo de carácter.

*argumento*

Un elemento, registro o matriz.

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

“PageHandler” en la página 194

#### **Consulta relacionada**

“Biblioteca J2EELib de EGL” en la página 802

“setRequestAttr()” en la página 804

#### **getSessionAttr()**

La función de sistema **J2EELib.getSessionAttr** utiliza una clave especificada para recuperar un argumento del objeto de sesión en una variable especificada. Esta función se utiliza en los PageHandlers y en los programas que se ejecutan en aplicaciones Web.

Si no se encuentra un objeto con la clave especificada, la variable destino no sufre cambios. Si el objeto recuperado es de un tipo incorrecto, se lanza una excepción y el programa o el PageHandler finalizan.

Puede colocar un argumento en el objeto de sesión mediante la función de sistema `J2EELib.setSessionAttr`.

```
J2EELib.getSessionAttr(
 clave STRING in,
 argumento attribute in)
```

*clave*

Un literal de carácter o un elemento de cualquier tipo de carácter.

*argumento*

Un elemento, registro o matriz.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“PageHandler” en la página 194

### Consulta relacionada

“Biblioteca J2EELib de EGL” en la página 802

“setSessionAttr()”

## setRequestAttr()

La función de sistema `J2EELib.setRequestAttr` utiliza una clave especificada para colocar un argumento especificado en el objeto de petición. Esta función se utiliza en los PageHandlers y en los programas que se ejecutan en aplicaciones Web.

Puede recuperar el argumento más tarde mediante la función de sistema

`J2EELib.getRequestAttr`.

```
J2EELib.setRequestAttr(
 clave STRING in,
 argumento attribute in)
```

*clave*

Un literal de carácter o un elemento de cualquier tipo de carácter.

*argumento*

Un elemento, registro o matriz.

En la salida Java generada, los argumentos de elemento se pasan como objetos primitivos Java (String, Integer, Decimal, etc.). Los argumentos de registro se pasan como beans de registro. Las matrices se pasan como lista de matrices del tipo asociado. El objeto de argumento se coloca en la colección de peticiones del servlet y está disponible para su acceso mientras la petición del servlet sea válida. Someter un formulario desde una página provoca la creación de una nueva petición.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“PageHandler” en la página 194

### Consulta relacionada

“Biblioteca J2EELib de EGL” en la página 802

“getRequestAttr()” en la página 803

## setSessionAttr()

La función de sistema `J2EELib.setSessionAttr` utiliza una clave especificada para colocar un argumento especificado en el objeto de sesión. Esta función se utiliza en

los PageHandlers y en los programas que se ejecutan en aplicaciones Web. Puede recuperar el argumento más tarde mediante la función de sistema J2EELib.getSessionAttr.

```
J2EELib.setSessionAttr(
 clave STRING in,
 argumento attribute in)
```

*clave*

Un literal de carácter o un elemento de cualquier tipo de carácter.

*argumento*

Un elemento, registro o matriz.

En la salida Java generada, los argumentos de elemento se pasan como objetos primitivos Java (String, Integer, Decimal, etc.). Los argumentos de registro se pasan como beans de registro. Las matrices se pasan como lista de matrices del tipo asociado.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“PageHandler” en la página 194

### Consulta relacionada

“Biblioteca J2EELib de EGL” en la página 802

“getSessionAttr()” en la página 803

## Biblioteca JavaLib de EGL

En la tabla figuran las funciones de acceso Java.

Función	Descripción
<i>result = getField (identifierOrClass, field)</i>	Devuelve el valor de un campo especificado de un objeto o clase especificados
<i>result = invoke (identifierOrClass, method[, argument])</i>	Invoca un método en un objeto o clase Java y puede devolver un valor
<i>result = isNull (identifier)</i>	Devuelve un valor (1 para true, 0 para false) para indicar si un identificador especificado hace referencia a un objeto nulo
<i>result = isObjID (identifier)</i>	Devuelve un valor (1 para true, 0 para false) para indicar si un identificador especificado se encuentra en el espacio de objetos
<i>result = qualifiedTypeName(identifier)</i>	Devuelve el nombre totalmente calificado de una clase de un objeto del espacio de objetos
<i>remove (identificador)</i>	Elimina el identificador especificado del espacio de objetos y, si ningún otro identificador hace referencia al objeto, lo elimina
<i>removeAll ()</i>	Elimina todos los identificadores y objetos del espacio de objetos
<i>setField (identificadorOClase, campo, valor)</i>	Establece el valor de un campo en un objeto o clase Java
<i>store (idAlmacén, identificadorOClase, método{,argumento})</i>	Invoca un método y coloca el objeto devuelto (o nulo) en el espacio de objetos, junto con un identificador especificado

Función	Descripción
storeCopy ( <i>idOrigen, idDestino</i> )	Crea un identificador nuevo basado en otro en el espacio de objetos, para que ambos hagan referencia al mismo objeto
storeField ( <i>idAlmacén, identificadorOClase, campo</i> )	Coloca el valor de un campo de clase o de objeto en el espacio de objetos
storeNew( <i>idAlmacén, clase{,argumento}</i> )	Invoca el constructor de una clase y coloca el objeto nuevo en el espacio de objetos

## Funciones de acceso Java

Las funciones de acceso *Java* son funciones de sistema EGL que permiten al código Java generado acceder a objetos y clases Java nativos; concretamente, para acceder a los métodos, constructores y campos de tipo público del código nativo.

Esta característica de EGL se hace posible durante la ejecución gracias a la presencia del *espacio de objetos Java EGL*, que es un conjunto de nombres y los objetos a los que estos nombres hacen referencia. Un único espacio de objetos está disponible en el programa generado y en todo el código Java generado al que el programa llama localmente, tanto si las llamadas son directas o a través de otro programa Java generado local, en cualquier nivel de la llamada. El espacio de objetos no está disponible en ningún código Java nativo.

Para almacenar y recuperar objetos del espacio de objetos, invoque las funciones de acceso Java. Las invocaciones incluyen el uso de identificadores, cada uno de los cuales es una serie que se utiliza para almacenar un objeto o para que coincida con un nombre que ya existe en el espacio de objetos. Cuando un identificador coincide con un nombre, el código puede acceder al objeto asociado al nombre.

Las secciones siguientes son estas:

- “Correlaciones de tipos EGL y Java”
- “Ejemplos” en la página 808
- “Manejo de errores” en la página 812

**Correlaciones de tipos EGL y Java:** Cada uno de los argumentos pasados a un método (y cada valor asignado a un campo) se correlaciona con un objeto o tipo primitivo Java. Los elementos del tipo primitivo EGL CHAR, por ejemplo, se pasan como objetos de la clase Java String. Se suministra un operador de conversión temporal para las situaciones en las que la correlación de tipos EGL con tipos Java no es suficiente.

Si se especifica un nombre Java, EGL elimina blancos de un solo byte y de doble byte del principio y el final del valor, que es sensible a mayúsculas y minúsculas. El truncamiento precede a cualquier conversión temporal. Este norma se aplica a los literales de serie y a los elementos de tipo CHAR, DBCHAR, MBCHAR o UNICODE. Este tipo de truncamiento no se produce si se especifica un argumento de método o valor de campo (por ejemplo, la serie “ mis datos ” se pasa a un método sin cambios), a menos que convierta temporalmente el valor a objID o nulo.

La tabla siguiente describe todas las correlaciones válidas.

Categoría de argumento		Ejemplos	Tipo Java
Un literal de serie o un elemento de tipo CHAR, DBCHAR, MBCHAR o UNICODE	Sin conversión temporal	"myString"	java.lang.String
	Conversión temporal con objId, que indica un identificador	(objId)"myId" x = "myId"; (objId)x	La clase del objeto al que el identificador hace referencia
	Conversión temporal con null, que puede ser adecuado para proporcionar una referencia nula a una clase totalmente calificada	(null)"java.lang.Thread" x = "java.util.HashMap"; (null)x	La clase especificada <b>Nota:</b> No puede pasarse una matriz convertida temporalmente con null, como por ejemplo (null)"int[]"
	Conversión temporal con char, que significa que se pasa el primer carácter del valor (cada ejemplo de la columna siguiente pasa una "a")	(char)"abc" x = "abc"; (char)x	char
Un elemento de tipo FLOAT o un literal de coma flotante	Sin conversión temporal	myFloatValue	double
Un elemento de tipo HEX	Sin conversión temporal	myHexValue	matriz de bytes
Un elemento de tipo SMALLFLOAT	Sin conversión temporal	mySmallFloat	float
Un elemento de tipo DATE	Sin conversión temporal	myDate	java.sql.Date
Un elemento de tipo TIME	Sin conversión temporal	myTime	java.sql.Time
Un elemento de tipo TIMESTAMP	Sin conversión temporal	myTimeStamp	java.sql.Timestamp
Un elemento de tipo INTERVAL	Sin conversión temporal	myInterval	java.lang.String
Literal de coma flotante	Sin conversión temporal	-6.5231E96	double

Categoría de argumento		Ejemplos	Tipo Java
Elemento numérico (o literal de coma no flotante) que no contiene decimales; los ceros iniciales se incluyen en el número de dígitos para un literal	Sin conversión temporal, 1-4 dígitos	0100	short
	Sin conversión temporal, 5-9 dígitos	00100	int
	Sin conversión temporal, 9-18 dígitos	1234567890	long
	Sin conversión temporal, > 18 dígitos	1234567890123456789	java.math.BigInteger
Elemento numérico (o literales de coma no flotante) que contiene decimales; los ceros iniciales y finales se incluyen en el número de dígitos para un literal	Sin conversión temporal, 1-6 dígitos	3.14159	float
	Sin conversión temporal, 7-18 dígitos	3.14159265	double
	Sin conversión temporal, > 18 dígitos	56789543.222	java.math.BigDecimal
Elemento o literal de coma no flotante, con o sin decimales	Conversión temporal con bigdecimal, biginteger, byte, double, float, short, int, long	X = 42; (byte)X (long)X	El tipo primitivo especificado; pero, si el valor está fuera de rango para ese tipo, se produce pérdida de precisión y el signo puede cambiar
	Conversión temporal con boolean, que significa que no cero es true, cero es false	X = 1; (boolean)X	boolean

**Nota:** Para evitar perder precisión, utilice un elemento float EGL para un double Java y un elemento smallfloat EGL para un float Java. Si utiliza otro de los tipos EGL, probablemente se redondeará un valor.

Para obtener detalles acerca del formato interno de los elementos en EGL, consulte las páginas de la ayuda referentes a *Tipos primitivos*.

**Ejemplos:** Esta sección ofrece ejemplos de utilización de funciones de acceso Java.

*Imprimir una serie de fecha:* El ejemplo siguiente imprime una serie de fecha:

```
// llamar al constructor de la clase Java Date y
// asignar el objeto nuevo al identificador "date".
JavaLib.storeNew((objId)"date", "java.util.Date");

// llamar al método toString del objeto nuevo Date
// y asignar la salida (fecha de hoy) a charItem.
// En ausencia de la conversión temporal (objId), "date"
// hace referencia a una clase en lugar de a un objeto.
charItem = JavaLib.invoke((objId)"date", "toString");
```

```

// asignar la corriente de salida estándar de la
// clase Java System al identificador "systemOut".
JavaLib.storeField((objId)"systemOut",
 "java.lang.System", "out");

// llamar al método println de la corriente de
// salida e imprimir la fecha de hoy.
JavaLib.invoke((objId)"systemOut","println",charItem);

// La utilización de "java.lang.System.out" como primer
// argumento en la línea anterior no habría sido
// válida, ya que el argumento debe ser un
// identificador que ya se encuentre en el espacio de objetos
// o un nombre de clase. El argumento no puede hacer referencia a un campo estático.

```

*Probar una propiedad del sistema:* El ejemplo siguiente recupera una propiedad del sistema y comprueba la ausencia de un valor:

```

// asignar el nombre de un identificador a un elemento de tipo CHAR
valueID = "osNameProperty"

// colocar el valor de la propiedad os.name en el
// espacio de objetos, y relacionar ese valor (una serie Java)
// con el identificador osNameProperty
JavaLib.store((objId)valueId, "java.lang.System",
 "getProperty", "os.name");

// comprobar si el valor de propiedad no existe
// y procesar de acuerdo con ello
myNullFlag = JavaLib.isNull((objId)valueId);

if(myNullFlag == 1)
 error = 27;
end

```

*Trabajar con matrices:* Al trabajar con matrices Java en EGL, utilice la clase Java `java.lang.reflect.Array`, como se muestra en los ejemplos posteriores y se describe en la documentación de API Java. No puede utilizar **JavaLib.storeNew** para crear una matriz Java, ya que las matrices Java no tienen constructores.

Debe utilizar el método estático `newInstance` de `java.lang.reflect.Array` para crear la matriz en el espacio de objetos. Una vez creada la matriz, utilice otros métodos de esa clase para acceder a los elementos.

El método `newInstance` espera dos argumentos:

- Un objeto `Class` que determina el tipo de matriz que se crea
- Un número que especifica cuántos elementos hay en la matriz

El código que identifica el objeto `Class` varía dependiendo de si se está creando una matriz de objetos o una matriz de primitivos. El código subsiguiente que interactúa con la matriz también varía sobre la misma base.

*Trabajar con una matriz de objetos:* El ejemplo siguiente muestra cómo crear una matriz de objetos de 5 elementos a la que puede accederse mediante el identificador "myArray":

```

// Obtener una referencia a la clase, para utilizarla con newInstance
JavaLib.store((objId)"objectClass", "java.lang.Class",
 "forName", "java.lang.Object");

```

```
// Crear la matriz en el espacio de objetos
JavaLib.store((objId)"myArray", "java.lang.reflect.Array",
 "newInstance", (objId)"objectClass", 5);
```

Si desea crear una matriz que contenga un tipo de objeto diferente, cambie el nombre de clase que se pasa a la primera invocación de **JavaLib.store**. Para crear una matriz de objetos String, por ejemplo, pase "java.lang.String" en lugar de "java.lang.Object".

Para acceder a un elemento de una matriz de objetos, utilice los métodos get y set de java.lang.reflect.Array. En el ejemplo siguiente, i y length son elementos numéricos:

```
length = JavaLib.invoke("java.lang.reflect.Array",
 "getLength", (objId)"myArray");
i = 0;

while (i < length)
 JavaLib.store((objId)"element", "java.lang.reflect.Array",
 "get", (objId)"myArray", i);

 // Aquí, procesar el elemento según convenga
 JavaLib.invoke("java.lang.reflect.Array", "set",
 (objId)"myArray", i, (objId)"element");
 i = i + 1;
end
```

El ejemplo anterior es equivalente al siguiente código Java:

```
int length = myArray.length;

for (int i = 0; i < length; i++)
{
 Object element = myArray[i];

 // Aquí, procesar el elemento según convenga

 myArray[i] = element;
}
```

*Trabajar con una matriz de primitivos Java:* Para crear una matriz que almacene un primitivo Java en lugar de un objeto, utilice un mecanismo diferente en los pasos que preceden a la utilización de java.lang.reflect.Array. En particular, obtenga el argumento Class para newInstance accediendo al campo estático TYPE de una clase de tipo primitivo.

Ej ejemplo siguiente crea myArray2, que es una matriz de enteros de 30 elementos:

```
// Obtener una referencia a la clase, para utilizarla con newInstance
JavaLib.storeField((objId)"intClass",
 "java.lang.Integer", "TYPE");

// Crear la matriz en el espacio de objetos
JavaLib.store((objId)"myArray2", "java.lang.reflect.Array",
 "newInstance", (objId)"intClass", 30);
```

Si desea crear una matriz que contenga un tipo de primitivo diferente, cambie el nombre de clase que se pasa a la primera invocación de **JavaLib.storeField**. Para crear una matriz de caracteres, por ejemplo, pase "java.lang.Character" en lugar de "java.lang.Integer".

Para acceder a un elemento de una matriz de primitivos, utilice los métodos de `java.lang.reflect.Array` específicos de un tipo primitivo. Tales métodos incluyen `getInt`, `setInt`, `getFloat`, `setFloat`, etc. En el ejemplo siguiente, `length`, `element` e `i` son elementos numéricos:

```
length = JavaLib.invoke("java.lang.reflect.Array",
 "getLength", (objId)"myArray2");
i = 0;

while (i < length)
 element = JavaLib.invoke("java.lang.reflect.Array",
 "getDouble", (objId)"myArray2", i);

 // Aquí, procesar un elemento según convenga

 JavaLib.invoke("java.lang.reflect.Array", "setDouble",
 (objId)"myArray2", i, element);
 i = i + 1;
end
```

El ejemplo anterior es equivalente al siguiente código Java:

```
int length = myArray2.length;

for (int i = 0; i < length; i++)
{
 double element = myArray2[i];

 // Aquí, procesar un elemento según convenga

 myArray2[i] = element;
}
```

*Trabajar con colecciones:* Para iterar una colección a la que hace referencia una variable denominada *list*, un programa Java hace lo siguiente:

```
Iterator contents = list.iterator();

while(contents.hasNext())
{
 Object myObject = contents.next();
 // Procesar myObject
}
```

Supongamos que `hasNext` contiene datos numéricos y que el programa ha relacionado una colección con un identificador denominado *list*. El siguiente código EGL es equivalente al código Java descrito anteriormente:

```
JavaLib.store((objId)"contents", (objId)"list", "iterator");
hasNext = JavaLib.invoke((objId)"contents", "hasNext");

while (hasNext == 1)
 JavaLib.store((objId)"myObject", (objId)"contents", "next");

 // Procesar myObject
 hasNext = JavaLib.invoke((objId)"contents", "hasNext");
end
```

*Convertir una matriz en una colección:* Para crear una colección a partir de una matriz de objetos, utilice el método `asList` de `java.util.Arrays`, como se muestra en el ejemplo siguiente:

```
// Crear una colección a partir de la matriz myArray
// y relacionar esa colección con el identificador "list
JavaLib.store((objId)"list", "java.util.Arrays",
 "asList", (objId)"myArray");
```

A continuación, iterar sobre list, como se muestra en la sección anterior.

La transferencia de una matriz a una colección sólo funciona con una matriz de objetos, no con una matriz de primitivos Java. tenga cuidado de no confundir `java.util.Arrays` con `java.lang.reflect.Array`.

**Manejo de errores:** Muchas de las funciones de acceso Java están asociadas con códigos de error, como se describe en las páginas de la ayuda específicas de las funciones. Si el valor de la variable de sistema `VGVar.handleSysLibraryErrors` es 1 cuando se produce uno de los errores indicados, EGL establece la variable de sistema `sysVar.errorCode` en un valor no cero. Si el valor de `VGVar.handleSysLibraryErrors` es 0 cuando se produce uno de los errores, el programa finaliza.

Es de particular interés el valor de `sysVar.errorCode` "00001000", que indica que un método invocado ha lanzado una excepción o como resultado de una inicialización de clase.

Cuando se lanza una excepción, EGL la almacena en el espacio de objetos. Si se produce otra excepción, la segunda ocupa el lugar de la primera. Puede utilizar el identificador `caughtException` para acceder a la última excepción producida.

En una situación inusual, un método invocado lanza no una excepción, sino un error como, por ejemplo, `OutOfMemoryError` o `StackOverflowError`. En tal caso, el programa finaliza independientemente del valor de la variable de sistema `VGVar.handleSysLibraryErrors`.

El siguiente código Java muestra cómo un programa Java puede tener varios bloques catch para manejar diferentes tipos de excepciones. Este código intenta crear un objeto `FileOutputStream`. Una anomalía provoca que el código establezca una variable `errorType` y que almacene la excepción que se ha lanzado.

```
int errorType = 0;
Exception ex = null;

try
{
 java.io.FileOutputStream fOut =
 new java.io.FileOutputStream("out.txt");
}
catch (java.io.IOException iox)
{
 errorType = 1;
 ex = iox;
}
catch (java.lang.SecurityException sx)
{
 errorType = 2;
 ex = sx;
}
```

El siguiente código EGL es equivalente al código Java anterior:

```
VGVar.handleSysLibraryErrors = 1;
errorType = 0;

JavaLib.storeNew((objId)"fOut",
 "java.io.FileOutputStream", "out.txt");

if (sysVar.errorCode == "00001000")
 exType = JavaLib.qualifiedTypeName((objId)"caughtException");
```

```

if (exType == "java.io.IOException")
 errorType = 1;
 JavaLib.storeCopy((objId)"caughtException", (objId)"ex");
else
 if (exType == "java.lang.SecurityException")
 errorType = 2;
 JavaLib.storeCopy((objId)"caughtException", (objId)"ex");
 end
end
end
end

```

### Consulta relacionada

“Biblioteca JavaLib de EGL” en la página 805

“Manejo de excepciones” en la página 94

“Tipos primitivos” en la página 34

“getField()”

“isNull()” en la página 818

“isObjID()” en la página 819

“qualifiedTypeName()” en la página 820 “remove()” en la página 821

“removeAll()” en la página 822

“setField()” en la página 823

“store()” en la página 824

“storeCopy()” en la página 826

“storeField()” en la página 827

“storeNew()” en la página 829

### getField()

La función de sistema **JavaLib.getField** devuelve el valor de un campo especificado de un objeto o clase especificados. La función **JavaLib.getField** es una de las diversas funciones de acceso Java.

```

JavaLib.getField(
 identificadorOClase javeObjIdOrClass in,
 campo STRING in)
returns (resultado anyJavaPrimitive)

```

*resultado*

El campo de resultado es obligatorio y recibe el valor del campo especificado en el segundo argumento. Se aplican las siguientes normas:

- Si el valor recibido es `BigDecimal`, `BigInteger`, `byte`, `short`, `int`, `long`, `float` o `double`, el campo de resultado debe ser un tipo de datos numérico. Las características no necesitan coincidir con el valor; por ejemplo, un tipo `float` puede almacenarse en una variable de retorno declarada sin dígitos decimales. Para obtener detalles acerca del manejo de desbordamiento, consulte `VGVar.handleOverflow` y `sysVar.overflowIndicator`.
- Si el valor recibido es de tipo booleano, el campo de resultado debe ser de tipo primitivo numérico. El valor es 1 para `true` y 0 para `false`.
- Si el valor recibido es una matriz de bytes, el campo de resultado debe ser de tipo HEX. Para obtener detalles acerca de la discrepancia de longitudes, consulte el tema *Asignaciones*.
- Si el valor recibido es un de tipo `String` o `char`, el campo de resultado debe ser de tipo `CHAR`, `DBCHAR`, `MBCHAR`, `STRING` o `UNICODE`--
  - Si el campo de resultado es de tipo `MBCHAR`, `STRING` o `UNICODE`, el valor recibido es siempre adecuado
  - Si el campo de resultado es de tipo `CHAR`, pueden producirse problemas si el valor recibido incluye caracteres que corresponden a caracteres `DBCHAR`

- Si el campo es de tipo DBCHAR, pueden producirse problemas si el valor recibido incluye caracteres Unicode que corresponden a caracteres de un solo byte

Para obtener detalles acerca de la discrepancia de longitudes, consulte el tema *Asignaciones*.

- Si el método nativo Java no devuelve ningún valor o devuelve un nulo, se produce el error 00001004, como se indica más adelante.

*identificadorOClase*

Este argumento es una de las siguientes entidades:

- Un identificador que hace referencia a un objeto del espacio de objetos; o
- El nombre totalmente calificado de una clase Java.

Este argumento es un literal de serie o una variable de tipo CHAR, DBCHAR, MBCHAR, STRING o UNICODE. Si especifica un identificador de un objeto, el identificador debe haberse convertido temporalmente a objID, como en el ejemplo que figura más adelante. Si tiene previsto especificar un campo estático en el siguiente argumento, es aconsejable especificar una clase en este argumento.

EGL elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

*campo*

El nombre del campo que debe leerse.

Este argumento es un literal de serie o una variable de tipo CHAR, DBCHAR, MBCHAR, STRING o UNICODE. Se eliminan blancos de un solo byte y de doble byte del principio y el final de la serie, que es sensible a mayúsculas y minúsculas.

A continuación se ofrece un ejemplo:

```
myVar = JavaLib.getField((objId)"myID", "myField");
```

Un error producido durante el proceso de **JavaLib.getField** puede establecer **sysVar.errorCode** en uno de los valores que figuran en la tabla siguiente.

Valor de sysVar.errorCode	Descripción
00001000	Un método invocado ha lanzado una excepción o como resultado de una inicialización de clase.
00001001	El objeto era nulo o el identificador especificado no estaba en el espacio de objetos
00001002	No existe o no puede cargarse un método, campo o clase públicos con el nombre especificado
00001004	El método ha devuelto nulo, no devuelve ningún valor o el valor de un campo era nulo
00001005	El valor devuelto no coincide con el tipo de la variable de retorno

Valor de sysVar.errorCode	Descripción
00001007	Se ha lanzado una excepción de tipo SecurityException o IllegalAccessException durante un intento de obtener información acerca de un método o campo; o se ha intentado establecer el valor de un campo declarado como final
00001009	Debe especificarse un identificador en lugar de un nombre de clase; el método o campo no es estático

## Conceptos relacionados

### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### Consulta relacionada

“Asignaciones” en la página 374

“BIN y los tipos enteros (integer)” en la página 50

“Biblioteca JavaLib de EGL” en la página 805

“Manejo de excepciones” en la página 94

“invoke()”

“isNull()” en la página 818

“isObjID()” en la página 819

“qualifiedTypeName()” en la página 820

“remove()” en la página 821

“removeAll()” en la página 822

“setField()” en la página 823

“store()” en la página 824

“storeCopy()” en la página 826

“storeField()” en la página 827

“storeNew()” en la página 829

## invoke()

La función de sistema **JavaLib.invoke** invoca un método en un objeto o clase Java nativa y puede devolver un valor. La función **JavaLib.invoke** es una de las diversas funciones de acceso Java.

```

JavaLib.invoke(
 identificadorOClase javaObjIdOrClass in,
 método STRING in
 {, argumento anyEglPrimitive in})
returns (resultado anyJavaPrimitive)

```

*resultado*

El campo de retorno, si está presente, recibe un valor del método nativo Java.

Si el método nativo Java devuelve un valor, el campo de resultado es opcional.

Se aplican las siguientes normas:

- Si el valor devuelto es BigDecimal, BigInteger, byte, short, int, long, float o double, el campo de resultado debe ser un tipo de datos numérico. Las características no necesitan coincidir con el valor; por ejemplo, un tipo float puede almacenarse en un campo de resultado declarado sin dígitos decimales. Para obtener detalles acerca del manejo del desbordamiento, consulte *VGVar.handleOverflow* y *SysVar.overflowIndicator*.

- Si el valor devuelto es de tipo booleano, el campo de resultado debe ser de tipo primitivo numérico. El valor es 1 para true y 0 para false.
- Si el valor devuelto es una matriz de bytes, el campo de resultado debe ser de tipo HEX. Para obtener detalles acerca de la discrepancia de longitudes, consulte el tema *Asignaciones*.
- Si el valor devuelto es un de tipo String o char, el campo de resultado debe ser de tipo CHAR, DBCHAR, MBCHAR, STRING o UNICODE--
  - Si el campo de resultado es de tipo MBCHAR, STRING o UNICODE, el valor devuelto es siempre adecuado
  - Si el campo de resultado es de tipo CHAR, pueden producirse problemas si el valor devuelto incluye caracteres que corresponden a caracteres DBCHAR
  - Si el campo es de tipo DBCHAR, pueden producirse problemas si el valor devuelto incluye caracteres Unicode que corresponden a caracteres de un solo byte

Para obtener detalles acerca de la discrepancia de longitudes, consulte el tema *Asignaciones*.
- Si el método nativo Java no devuelve ningún valor o devuelve un nulo, se aplican las siguientes normas:
  - No se produce ningún error en ausencia de un campo de resultado
  - Se produce un error durante la ejecución si está presente un campo de resultado; el error es el 00001004, como se indica más adelante

#### *identificadorOClase*

Este argumento es una de las siguientes entidades:

- Un identificador que hace referencia a un objeto del espacio de objetos; o
- El nombre totalmente calificado de una clase Java.

Este argumento es un literal de serie o una variable de tipo CHAR, DBCHAR, MBCHAR, STRING o UNICODE. Si especifica un identificador de un objeto, el identificador debe haberse convertido temporalmente a objID, como en el ejemplo que figura más adelante. EGL elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

El código no puede invocar un método en un objeto hasta que el usuario ha creado un identificador para el objeto. Un ejemplo que figura más adelante ilustra este punto con `java.lang.System.out`, que hace referencia a un objeto `PrintStream`.

#### *método*

El nombre del método al que debe llamarse.

Este argumento es un literal de serie o una variable de tipo CHAR, DBCHAR, MBCHAR, STRING o UNICODE. Se eliminan blancos de un solo byte y de doble byte del principio y el final de la serie, que es sensible a mayúsculas y minúsculas.

#### *argumento*

Un valor pasado al método.

Puede ser necesaria una conversión temporal, como se especifica en el apartado *Acceso Java (palabras del sistema)*.

Se aplican las normas de conversión de tipos Java. Por ejemplo, no se produce ningún error si se pasa un tipo short a un parámetro de método declarado como int.

Para evitar perder precisión, utilice un campo float EGL para un double Java y un campo smallfloat EGL para un float Java. Si utiliza otro de los tipos EGL, probablemente se redondeará un valor.

El área de memoria del programa invocante no cambia, independientemente de la acción realizada por el método.

En el ejemplo siguiente, es necesaria la conversión temporal (objId) excepto cuando se indica:

```
// llamar al constructor de la clase Java Date y
// asignar el objeto nuevo al identificador "date".
JavaLib.storeNew((objId)"date", "java.util.Date");

// llamar al método toString del objeto nuevo Date
// y asignar la salida (fecha de hoy) a chaItem.
// En ausencia de la conversión temporal (objId), "date"
// hace referencia a una clase en lugar de a un objeto.
chaItem = JavaLib.invoke((objId)"date", "toString");

// asignar la corriente de salida estándar de la
// clase Java System al identificador "systemOut".
JavaLib.storeField((objId)"systemOut", "java.lang.System", "out");

// llamar al método println de la corriente de
// salida e imprimir la fecha de hoy.
JavaLib.invoke((objID)"systemOut", "println", chaItem);

// La utilización de "java.lang.System.out" como primer
// argumento en la línea anterior no habría sido
// válida, ya que el argumento debe ser un
// identificador que ya se encuentre en el espacio de objetos
// o un nombre de clase. El argumento no puede hacer referencia a un campo estático.
```

Un error producido durante el proceso de **JavaLib.invoke** puede establecer **SysVar.errorCode** en uno de los valores que figuran en la tabla siguiente.

Valor de SysVar.errorCode	Descripción
00001000	Un método invocado ha lanzado una excepción o como resultado de una inicialización de clase.
00001001	El objeto era nulo o el identificador especificado no estaba en el espacio de objetos
00001002	No existe o no puede cargarse un método, campo o clase públicos con el nombre especificado
00001003	El tipo primitivo EGL no coincide con el tipo esperado en Java
00001004	El método ha devuelto nulo, no devuelve ningún valor o el valor de un campo era nulo
00001005	El valor devuelto no coincide con el tipo de la variable de retorno
00001006	No ha podido cargarse la clase de una conversión temporal de argumento a nulo

Valor de SysVar.errorCode	Descripción
00001007	Se ha lanzado una excepción de tipo SecurityException o IllegalAccessException durante un intento de obtener información acerca de un método o campo; o se ha intentado establecer el valor de un campo declarado como final
00001009	Debe especificarse un identificador en lugar de un nombre de clase; el método o campo no es estático

## Conceptos relacionados

### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### Consulta relacionada

“Asignaciones” en la página 374

“BIN y los tipos enteros (integer)” en la página 50

“Biblioteca JavaLib de EGL” en la página 805

“Manejo de excepciones” en la página 94

“getField()” en la página 813

“isNull()”

“isObjID()” en la página 819

“qualifiedTypeName()” en la página 820

“remove()” en la página 821

“removeAll()” en la página 822

“setField()” en la página 823

“store()” en la página 824

“storeCopy()” en la página 826

“storeField()” en la página 827

“storeNew()” en la página 829

“Tipos primitivos” en la página 34

“overflowIndicator” en la página 932

“handleOverflow” en la página 946

## isNull()

La función de sistema **JavaLib.isNull** devuelve un valor (1 para true, 0 para false) para indicar si un identificador especificado hace referencia a un objeto nulo. La función **JavaLib.isNull** es una de las diversas funciones de acceso Java.

```
JavaLib.isNull(identificador javaObjId in)
returns (resultado INT)
```

### *resultado*

Un campo numérico que recibe uno de dos valores: 1 para true, 0 para false. La utilización de un elemento no numérico provoca un error durante la validación.

### *identificador*

Un identificador que hace referencia a un objeto del espacio de objetos.

Este argumento es un literal de serie o un elemento de tipo CHAR, DBCHAR, MBCHAR, STRING o UNICODE. El identificador debe haberse convertido

temporalmente a objID. EGL elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

A continuación se ofrece un ejemplo:

```
// probar si un objeto es nulo
// y procesar de acuerdo con ello
isNull = JavaLib.isNull((objId)valueId);

if(isNull == 1)
 error = 12;
end
```

Un error producido durante el proceso de **JavaLib.isNull** puede establecer **SysVar.errorCode** en uno de los valores que figuran en la tabla siguiente.

Valor de sysVar.errorCode	Descripción
00001001	El identificador especificado no estaba en el espacio de objetos

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca JavaLib de EGL” en la página 805

“getField()” en la página 813

“invoke()” en la página 815

“isObjID()”

“qualifiedTypeName()” en la página 820

“remove()” en la página 821

“removeAll()” en la página 822

“setField()” en la página 823

“store()” en la página 824

“storeCopy()” en la página 826

“storeField()” en la página 827

“storeNew()” en la página 829

### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

## isObjID()

La función de sistema **JavaLib.isObjID** devuelve un valor (1 para true, 0 para false) para indicar si un identificador especificado se encuentra en el espacio de objetos. La función **JavaLib.isObjID** es una de las diversas funciones de acceso Java.

```
JavaLib.isObjID(identificador javaObjId in)
returns (resultado INT)
```

*resultado*

Un elemento numérico que recibe un de dos valores: 1 para true, 0 para false.

La utilización de un elemento no numérico provoca un error durante la validación.

*identificador*

Un identificador que hace referencia a un objeto del espacio de objetos.

Este argumento es un literal de serie o un elemento de tipo CHAR, DBCHAR, MBCHAR o UNICODE. El identificador debe haberse convertido temporalmente a objID. EGL elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

A continuación se ofrece un ejemplo:

```
// probar si un objeto no existe
// y procesar de acuerdo con ello
isPresent = JavaLib.isObjID((objId)valueId);

if(isPresent == 0)
 error = 27;
end
```

No existen errores de tiempo de ejecución asociados con **JavaLib.isObjID**.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“Funciones de acceso Java” en la página 806

### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### Consulta relacionada

“Biblioteca JavaLib de EGL” en la página 805

“getField()” en la página 813

“invoke()” en la página 815

“isNull()” en la página 818

“qualifiedTypeName()”

“remove()” en la página 821

“removeAll()” en la página 822

“setField()” en la página 823

“store()” en la página 824

“storeCopy()” en la página 826

“storeField()” en la página 827

“storeNew()” en la página 829

### qualifiedTypeName()

La función de sistema **JavaLib.qualifiedTypeName** devuelve el nombre totalmente calificado de la clase de un objeto del espacio de objetos Java de EGL. La función **JavaLib.qualifiedTypeName** es una de las diversas funciones de acceso Java.

```
JavaLib.qualifiedTypeName(identificador javaObjId in)
returns (resultado STRING)
```

*resultado*

El campo de resultado es obligatorio y debe ser de tipo CHAR, MBCHAR o UNICODE:

- Si el campo es de tipo MBCHAR o UNICODE, el valor recibido siempre es correcto
- Si el campo de resultado es de tipo CHAR, pueden producirse problemas si el valor recibido incluye caracteres que corresponden a caracteres DBCHAR

Para obtener detalles acerca de la discrepancia de longitudes, consulte el tema *Asignaciones*.

*identificador*

Un identificador que hace referencia a un objeto del espacio de objetos.

Este argumento es un literal de serie o un elemento de tipo CHAR, DBCHAR, MBCHAR o UNICODE. El identificador debe haberse convertido temporalmente a objId, como en el ejemplo que figura más adelante. EGL elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

A continuación se ofrece un ejemplo:

```
myItem = JavaLib.qualifiedTypeName((objId)"myId");
```

Un error producido durante el proceso de **JavaLib.qualifiedTypeName** puede establecer **sysVar.errorCode** en uno de los valores que figuran en la tabla siguiente.

Valor de sysVar.errorCode	Descripción
00001001	El objeto era nulo o el identificador especificado no estaba en el espacio de objetos

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca JavaLib de EGL” en la página 805

“getField()” en la página 813

“invoke()” en la página 815

“isNull()” en la página 818

“isObjID()” en la página 819

“qualifiedTypeName()” en la página 820

“remove()”

“removeAll()” en la página 822

“setField()” en la página 823

“store()” en la página 824

“storeCopy()” en la página 826

“storeField()” en la página 827

“storeNew()” en la página 829

### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

## remove()

La función de sistema **JavaLib.remove** elimina el identificador especificado del espacio de objetos Java de EGL. El objeto relacionado con el identificador también se elimina, pero sólo si el identificador es el único que hace referencia al objeto. Si otro identificador hace referencia al objeto, éste permanece en el espacio de objetos y es accesible por medio de ese otro identificador.

La función **JavaLib.remove** es una de las diversas funciones de acceso Java.

```
JavaLib.remove(identificador javaObjId in)
```

*identificador*

El identificador que hace referencia a un objeto. No se produce ningún error si no se encuentra el identificador.

Este argumento es un literal de serie o una variable de tipo CHAR, DBCHAR, MBCHAR, STRING o UNICODE. El identificador debe haberse convertido temporalmente a objID, como en el ejemplo que figura más adelante. EGL

elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

A continuación se ofrece un ejemplo:

```
JavaLib.remove((objId)myStoredObject);
```

No existen errores de tiempo de ejecución asociados con **JavaLib.remove**.

**Nota:** Al invocar las funciones de sistema **JavaLib.remove** y **JavaLib.removeAll**, el código permite que la máquina virtual Java maneje la recogida de basura del espacio de objetos Java de EGL. Si no invoca una función de sistema para eliminar un objeto del espacio de objetos, la memoria no se recupera durante la ejecución de cualquier programa que tenga acceso al espacio de objetos.

### Conceptos relacionados

#### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

#### Consulta relacionada

“Biblioteca JavaLib de EGL” en la página 805

“getField()” en la página 813

“invoke()” en la página 815

“isNull()” en la página 818

“isObjID()” en la página 819

“qualifiedTypeName()” en la página 820

“removeAll()”

“setField()” en la página 823

“store()” en la página 824

“storeCopy()” en la página 826

“storeField()” en la página 827

“storeNew()” en la página 829

### removeAll()

La función de sistema **JavaLib.removeAll** elimina todos los identificadores y objetos del espacio de objetos Java de EGL. La función **JavaLib.removeAll** es una de las diversas funciones de acceso Java.

```
JavaLib.removeAll()
```

No existen errores de tiempo de ejecución asociados con **JavaLib.removeAll**.

**Nota:** Al invocar las funciones de sistema **JavaLib.remove** y **JavaLib.removeAll**, el código permite que la máquina virtual Java maneje la recogida de basura del espacio de objetos Java de EGL. Si no invoca una función de sistema para eliminar un objeto del espacio de objetos, la memoria no se recupera durante la ejecución de cualquier programa que tenga acceso al espacio de objetos.

### Conceptos relacionados

#### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

## Consulta relacionada

"Biblioteca JavaLib de EGL" en la página 805

"getField()" en la página 813  
"invoke()" en la página 815  
"isNull()" en la página 818  
"isObjID()" en la página 819  
"qualifiedTypeName()" en la página 820  
"remove()" en la página 821  
"setField()" en la página 824  
"storeCopy()" en la página 826  
"storeField()" en la página 827  
"storeNew()" en la página 829

## setField()

La función de sistema **JavaLib.setField** establece el valor de un campo de un objeto o clase nativa Java. La función **JavaLib.setField** es una de las diversas funciones de acceso Java.

```
JavaLib.setField(
 identificadorOClase javaObjId in,
 campo STRING in,
 valor anyEglPrimitive in)
```

*identificadorOClase*

Este argumento es una de las siguientes entidades:

- Un identificador que hace referencia a un objeto del espacio de objetos; o
- El nombre totalmente calificado de una clase Java.

Este argumento es un literal de serie o una variable de tipo CHAR, DBCHAR, MBCHAR, STRING o UNICODE. Si especifica un identificador de un objeto, el identificador debe haberse convertido temporalmente a objID, como en el ejemplo que figura más adelante. EGL elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

*campo*

El nombre del campo que debe cambiarse.

Este argumento es un literal de serie o una variable de tipo CHAR, DBCHAR, MBCHAR, STRING o UNICODE. Se eliminan blancos de un solo byte y de doble byte del principio y el final de la serie, que es sensible a mayúsculas y minúsculas.

*valor*

El valor propiamente dicho.

Puede ser necesaria una conversión temporal, como se especifica en el apartado Acceso Java (palabras del sistema).

Se aplican las normas de conversión de tipos Java. Por ejemplo, no se produce ningún error si se asigna un tipo short a un campo declarado como int.

A continuación se ofrece un ejemplo:

```
JavaLib.setField((objID)"myId", "myField",
 (short)myNumItem);
```

Un error producido durante el proceso de **JavaLib.setField** puede establecer **SysVar.errorCode** en uno de los valores que figuran en la tabla siguiente.

Valor de SysVar.errorCode	Descripción
00001000	Un método invocado ha lanzado una excepción o como resultado de una inicialización de clase.
00001001	El objeto era nulo o el identificador especificado no estaba en el espacio de objetos
00001002	No existe o no puede cargarse un método, campo o clase públicos con el nombre especificado
00001003	El tipo primitivo EGL no coincide con el tipo esperado en Java
00001007	Se ha lanzado una excepción de tipo <code>SecurityException</code> o <code>IllegalAccessException</code> durante un intento de obtener información acerca de un método o campo; o se ha intentado establecer el valor de un campo declarado como final
00001009	Debe especificarse un identificador en lugar de un nombre de clase; el método o campo no es estático

### Conceptos relacionados

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### Consulta relacionada

“Biblioteca JavaLib de EGL” en la página 805

“`getField()`” en la página 813

“`invoke()`” en la página 815

“`isNull()`” en la página 818

“`isObjID()`” en la página 819

“`qualifiedTypeName()`” en la página 820

“`remove()`” en la página 821

“`removeAll()`” en la página 822

“`store()`”

“`storeCopy()`” en la página 826

“`storeField()`” en la página 827

“`storeNew()`” en la página 829

### store()

La función de sistema **JavaLib.store** invoca un método y coloca el objeto devuelto (o nulo) en el espacio de objetos Java de EGL, junto con un identificador especificado. Si el identificador ya se encuentra en el espacio de objetos, la acción es equivalente a los siguientes pasos:

- Ejecutar **JavaLib.remove** en el identificador para eliminar el objeto relacionado con ese identificador
- Relacionar el objeto devuelto por **JavaLib.store** con el identificador destino

Si el método devuelve un primitivo Java en lugar de un objeto, EGL almacena un objeto que representa el primitivo; por ejemplo, si el método devuelve un int, EGL almacena un objeto de tipo java.lang.Integer.

La función **JavaLib.store** es una de las diversas funciones de acceso Java.

```
JavaLib.store(
 idAlmacén javaObjId in,
 identificadorOClase javaObjId in,
 método STRING in
 {, argumento anyEglPrimitive in})
```

*idAlmacén*

El identificador que debe almacenarse con el objeto devuelto.

Este argumento es un literal de serie o una variable de tipo CHAR, DBCHAR, MBCHAR, STRING o UNICODE. El identificador debe haberse convertido temporalmente a objID, como en el ejemplo que figura más adelante. EGL elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

*identificadorOClase*

Este argumento es una de las siguientes entidades:

- Un identificador que hace referencia a un objeto del espacio de objetos; o
- El nombre totalmente calificado de una clase Java.

Este argumento es un literal de serie o un elemento de tipo CHAR, DBCHAR, MBCHAR, STRING o UNICODE. Si especifica un identificador de un objeto, el identificador debe haberse convertido temporalmente a objID, como en el ejemplo que figura más adelante. EGL elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

*método*

El método que debe invocarse.

Este argumento es un literal de serie o una variable de tipo CHAR, DBCHAR, MBCHAR, STRING o UNICODE. Si especifica un identificador de un objeto, el identificador debe haberse convertido temporalmente a objID, como en el ejemplo que figura más adelante. EGL elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

*argumento*

Un valor pasado al método.

Puede ser necesaria una conversión temporal, como se especifica en el apartado Acceso Java (palabras del sistema).

Se aplican las normas de conversión de tipos Java. Por ejemplo, no se produce ningún error si se pasa un tipo short a un parámetro de método declarado como int.

Para evitar perder precisión, utilice un elemento float EGL para un double Java y un elemento smallfloat EGL para un float Java. Si utiliza otro de los tipos EGL, probablemente se redondeará un valor.

El área de memoria del programa invocante no cambia, independientemente de la acción realizada por el método.

A continuación se ofrece un ejemplo:

```
JavaLib.store((objId)"storeId", (objId)"myId",
"myMethod", 36);
```

Un error producido durante el proceso de **JavaLib.store** puede establecer **sysVar.errorCode** en uno de los valores que figuran en la tabla siguiente.

Valor de sysVar.errorCode	Descripción
00001000	Un método invocado ha lanzado una excepción o como resultado de una inicialización de clase.
00001001	El objeto era nulo o el identificador especificado no estaba en el espacio de objetos
00001002	No existe o no puede cargarse un método, campo o clase públicos con el nombre especificado
00001003	El tipo primitivo EGL no coincide con el tipo esperado en Java
00001006	No ha podido cargarse la clase de una conversión temporal de argumento a nulo
00001007	Se ha lanzado una excepción de tipo SecurityException o IllegalAccessException durante un intento de obtener información acerca de un método o campo; o se ha intentado establecer el valor de un campo declarado como final
00001009	Debe especificarse un identificador en lugar de un nombre de clase; el método o campo no es estático

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca JavaLib de EGL” en la página 805

“getField()” en la página 813

“invoke()” en la página 815

“isNull()” en la página 818

“isObjID()” en la página 819

“qualifiedTypeName()” en la página 820

“remove()” en la página 821

“removeAll()” en la página 822

“setField()” en la página 823

“storeCopy()”

“storeField()” en la página 827

“storeNew()” en la página 829

### storeCopy()

La función de sistema **JavaLib.storeCopy** crea un identificador nuevo basado en otro en el espacio de objetos, para que ambos hagan referencia al mismo objeto. Si el identificador origen no se encuentra en el espacio de objetos, se almacena un nulo para el identificador destino y no se produce ningún error. Si el identificador destino ya se encuentra en el espacio de objetos, la acción es equivalente a los siguientes pasos:

- Ejecutar **JavaLib.remove** en el identificador destino para eliminar el objeto relacionado con ese identificador
- Relacionar el objeto origen con el identificador destino

La función **JavaLib.storeCopy** es una de las diversas funciones de acceso Java.

```
JavaLib.storeCopy(
 idOrigen javaObjId in,
 idDestino javaObjId in)
```

*idOrigen*

Un identificador que hace referencia a un objeto del espacio de objetos o a nulo.

Este argumento es un literal de serie o una variable de tipo CHAR, DBCHAR, MBCHAR, STRING o UNICODE. El identificador debe haberse convertido temporalmente a objId, como en el ejemplo que figura más adelante. EGL elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

*idDestino*

El identificador nuevo, que hace referencia al mismo objeto.

Este argumento es un literal de serie o un elemento de tipo CHAR, DBCHAR, MBCHAR, STRING o UNICODE. El identificador debe haberse convertido temporalmente a objID, como en el ejemplo que figura más adelante. EGL elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

A continuación se ofrece un ejemplo:

```
JavaLib.storeCopy((objId)"sourceId", (objId)"targetId");
```

No existen errores de tiempo de ejecución asociados con **JavaLib.storeCopy**.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca JavaLib de EGL” en la página 805

“getField()” en la página 813

“invoke()” en la página 815

“isNull()” en la página 818

“isObjID()” en la página 819

“qualifiedTypeName()” en la página 820

“remove()” en la página 821

“removeAll()” en la página 822

“setField()” en la página 823

“store()” en la página 824

“storeField()”

“storeNew()” en la página 829

### storeField()

La función de sistema **JavaLib.storeField** coloca el valor de un campo de clase o de un campo de objeto en el espacio de objetos de Java. Si el identificador utilizado para almacenar el objeto ya se encuentra en el espacio de objetos, la acción es equivalente a los siguientes pasos:

- Ejecutar **JavaLib.remove** en el identificador para eliminar el objeto relacionado con el identificador

- Relacionar el objeto nuevo con el identificador

Si el campo de clase u objeto contiene un primitivo Java en lugar de un objeto, EGL almacena un objeto que representa el primitivo; por ejemplo, si el campo devuelve un int, EGL almacena un objeto de tipo java.lang.Integer.

```
JavaLib.storeField(
 idAlmacén javaObjId in,
 identificadorOClase javaObjIdOrClass in,
 campo STRING in)
```

*idAlmacén*

El identificador que debe almacenarse con el objeto.

Este argumento es un literal de serie o un elemento de tipo CHAR, DBCHAR, MBCHAR o UNICODE. El identificador debe haberse convertido temporalmente a objID, como en el ejemplo que figura más adelante. EGL elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

*identificadorOClase*

Este argumento es una de las siguientes entidades:

- Un identificador que hace referencia a un objeto del espacio de objetos; o
- El nombre totalmente calificado de una clase Java.

Este argumento es un literal de serie o un elemento de tipo CHAR, DBCHAR, MBCHAR o UNICODE. Si especifica un identificador de un objeto, el identificador debe haberse convertido temporalmente a objID, como en el ejemplo que figura más adelante. Si tiene previsto especificar un campo estático en el siguiente argumento, es aconsejable especificar una clase en este argumento.

EGL elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

*campo*

El nombre del campo que hace referencia a un objeto.

Este argumento es un literal de serie o un elemento de tipo CHAR, DBCHAR, MBCHAR o UNICODE. Se eliminan blancos de un solo byte y de doble byte del principio y el final de la serie, que es sensible a mayúsculas y minúsculas.

A continuación se ofrece un ejemplo:

```
JavaLib.storeField((objId)"myStoreId",
 (objId)"myId", "myField");
```

Un error producido durante el proceso de **JavaLib.storeField** puede establecer **sysVar.errorCode** en uno de los valores que figuran en la tabla siguiente.

Valor de sysVar.errorCode	Descripción
00001000	Un método invocado ha lanzado una excepción o como resultado de una inicialización de clase.
00001001	El objeto era nulo o el identificador especificado no estaba en el espacio de objetos

Valor de sysVar.errorCode	Descripción
00001002	No existe o no puede cargarse un método, campo o clase públicos con el nombre especificado
00001007	Se ha lanzado una excepción de tipo SecurityException o IllegalAccessException durante un intento de obtener información acerca de un método o campo; o se ha intentado establecer el valor de un campo declarado como final
00001009	Debe especificarse un identificador en lugar de un nombre de clase; el método o campo no es estático

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca JavaLib de EGL” en la página 805

“getField()” en la página 813

“invoke()” en la página 815

“isNull()” en la página 818

“isObjID()” en la página 819

“qualifiedTypeName()” en la página 820

“remove()” en la página 821

“removeAll()” en la página 822

“setField()” en la página 823

“store()” en la página 824

“storeCopy()” en la página 826

“storeNew()”

### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### storeNew()

La función de sistema **JavaLib.storeNew** invoca el constructor de una clase y coloca el objeto nuevo en el espacio de objetos Java de EGL. Si el identificador ya se encuentra en el espacio de objetos, la acción es equivalente a los siguientes pasos:

- Ejecutar **JavaLib.remove** en el identificador para eliminar el objeto relacionado anteriormente con el identificador
- Relacionar el objeto nuevo con el identificador

La función **JavaLib.storeNew** es una de las diversas funciones de acceso Java.

```
JavaLib.storeNew(
 idAlmacén javaObjId in,
 clase STRING in
 {, argumento anyEglPrimitive in})
```

*idAlmacén*

El identificador que debe almacenarse con el objeto nuevo.

Este argumento es un literal de serie o un elemento de tipo CHAR, DBCHAR, MBCHAR o UNICODE. El identificador debe haberse convertido temporalmente a objID, como en el ejemplo que figura más adelante. EGL

elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

*clase*

El nombre totalmente calificado de una clase Java.

Este argumento es un literal de serie o un elemento de tipo CHAR, DBCHAR, MBCHAR o UNICODE. EGL elimina blancos de un solo byte y de doble byte del principio y el final del valor del argumento, que es sensible a mayúsculas y minúsculas.

*argumento*

Un valor pasado al constructor.

Puede ser necesaria una conversión temporal, como se especifica en el apartado *Acceso Java (palabras del sistema)*.

Se aplican las normas de conversión de tipos Java. Por ejemplo, no se produce ningún error si se pasa un tipo short a un parámetro de constructor declarado como int.

Para evitar perder precisión, utilice un elemento float EGL para un double Java y un elemento smallfloat EGL para un float Java. Si utiliza otro de los tipos EGL, probablemente se redondeará un valor.

El área de memoria del programa invocante no cambia, independientemente de la acción realizada por el constructor.

A continuación se ofrece un ejemplo:

```
JavaLib.storeNew((objId)"storeId", "myClass", 36);
```

Un error producido durante el proceso de **JavaLib.storeNew** puede establecer **sysVar.errorCode** en uno de los valores que figuran en la tabla siguiente.

Valor de sysVar.errorCode	Descripción
00001000	Un método invocado ha lanzado una excepción o como resultado de una inicialización de clase.
00001001	El objeto era nulo o el identificador especificado no estaba en el espacio de objetos
00001002	No existe o no puede cargarse un método, campo o clase públicos con el nombre especificado
00001003	El tipo primitivo EGL no coincide con el tipo esperado en Java
00001006	No ha podido cargarse la clase de una conversión temporal de argumento a nulo
00001007	Se ha lanzado una excepción de tipo SecurityException o IllegalAccessException durante un intento de obtener información acerca de un método o campo; o se ha intentado establecer el valor de un campo declarado como final
00001008	No puede llamarse al constructor; el nombre de clase hace referencia a una clase abstracta o interfaz

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca JavaLib de EGL” en la página 805

“getField()” en la página 813

“invoke()” en la página 815

“isNull()” en la página 818

“isObjID()” en la página 819

“qualifiedTypeName()” en la página 820

“remove()” en la página 821

“removeAll()” en la página 822

“setField()” en la página 823

“store()” en la página 824

“storeCopy()” en la página 826

“storeField()” en la página 827

### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

## Biblioteca LobLib de EGL

La tabla siguiente lista las funciones de la biblioteca LobLib.

Función de sistema/invocación	Descripción
<code>attachBlobToFile(variableBlob, nombreArchivo)</code>	Copia los datos a los que hace referencia una variable de tipo BLOB en un archivo especificado.
<code>attachBlobToTempFile(variableBlob )</code>	Copia los datos a los que hace referencia una variable de tipo BLOB en un sistema de archivos temporal exclusivo.
<code>attachClobToFile(variableClob, nombreArchivo)</code>	Copia los datos a los que hace referencia una variable de tipo CLOB en un archivo especificado.
<code>attachClobToTempFile(variableClob )</code>	Copia los datos a los que hace referencia una variable de tipo CLOB en un sistema de archivos temporal exclusivo.
<code>freeBlob(variableBlob)</code>	Libera los recursos utilizados por una variable de tipo BLOB.
<code>freeClob(variableClob)</code>	Libera los recursos utilizados por una variable de tipo CLOB.
<code>result = getBlobLen(blobVariable )</code>	Devuelve el número de bytes en el valor al que hace referencia una variable de tipo BLOB.
<code>result = getClobLen(clobVariable)</code>	Devuelve el número de caracteres a los que hace referencia una variable de tipo CLOB.
<code>result = getStrFromClob(clobVariable)</code>	Devuelve una serie que corresponde al valor al que hace referencia una variable de tipo CLOB.
<code>result = getSubStrFromClob(clobVariable, pos, longitud)</code>	Devuelve una subserie del valor al que hace referencia una variable de tipo CLOB.

Función de sistema/invocación	Descripción
<code>loadBlobFromFile(variableBlob, nombreArchivo)</code>	Copia los datos de un archivo especificado en un área de memoria a la que hace referencia una variable de tipo BLOB.
<code>loadClobFromFile(variableClob, nombreArchivo)</code>	Copia los datos de un archivo especificado en un área de memoria a la que hace referencia una variable de tipo CLOB.
<code>setClobFromString(variableClob, serie)</code>	Copia una serie en un área de memoria a la que hace referencia una variable de tipo CLOB.
<code>setClobFromStringAtPosition(variableClob, pos, serie)</code>	Copia una serie en un área de memoria a la que hace referencia una variable de tipo CLOB, empezando en una posición especificada del área de memoria.
<code>truncateBlob(variableBlob, longitud)</code>	Trunca el valor al que hace referencia una variable de tipo BLOB.
<code>truncateClob(variableClob, longitud)</code>	Trunca el valor al que hace referencia una variable de tipo CLOB.
<code>updateBlobToFile(variableBlob, nombreArchivo)</code>	Copia los datos a los que hace referencia una variable de tipo BLOB en un archivo especificado.
<code>updateClobToFile(variableClob, nombreArchivo)</code>	Copia los datos a los que hace referencia una variable de tipo CLOB en un archivo especificado.

## attachBlobToFile()

La función de sistema **LobLib.attachBlobToFile** copia los datos a los que hace referencia una variable de tipo BLOB en un archivo especificado.

```
LobLib.attachBlobToFile(
 variableBlob BLOB inOut,
 nombreArchivo STRING in)
```

*variableBlob*

La variable de tipo BLOB.

*nombreArchivo*

El nombre del archivo. El nombre está totalmente calificado o es relativo al directorio desde el que se invoca el programa.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“BLOB” en la página 49

“Biblioteca LobLib de EGL” en la página 831

## attachBlobToTempFile()

La función de sistema **LobLib.attachBlobToTempFile** copia los datos a los que hace referencia una variable de tipo BLOB en un sistema de archivos temporal exclusivo. Esta función minimiza la memoria utilizada en tiempo de ejecución.

```
LobLib.attachBlobToTempFile(variableBlob BLOB in)
```

*variableBlob*

La variable de tipo BLOB.

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“BLOB” en la página 49

“Biblioteca LobLib de EGL” en la página 831

### **attachClobToFile()**

La función de sistema **LobLib.attachClobToFile** copia los datos a los que hace referencia una variable de tipo CLOB en un archivo especificado.

```
LobLib.attachClobToFile(
 variableClob CLOB inOut,
 nombreArchivo STRING in)
```

*variableClob*

La variable de tipo CLOB.

*nombreArchivo*

El nombre del archivo. El nombre está totalmente calificado o es relativo al directorio desde el que se invoca el programa.

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“CLOB” en la página 48

“Biblioteca LobLib de EGL” en la página 831

### **attachClobToTempFile()**

La función de sistema **LobLib.attachClobToTempFile** copia los datos a los que hace referencia una variable de tipo CLOB en un sistema de archivos temporal exclusivo. Esta función minimiza la memoria utilizada en tiempo de ejecución.

```
LobLib.attachClobToTempFile(variableClob CLOB in)
```

*variableClob*

La variable de tipo CLOB.

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“CLOB” en la página 48

“Biblioteca LobLib de EGL” en la página 831

### **freeBlob()**

La función del sistema **LobLib.freeBlob** libera los recursos utilizados por una variable de tipo BLOB.

```
LobLib.freeBlob(variableBlob BLOB inOut)
```

*variableBlob*

La variable de tipo BLOB.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“BLOB” en la página 49

“Biblioteca LobLib de EGL” en la página 831

## freeClob()

La función del sistema **LobLib.freeClob** libera los recursos utilizados por una variable de tipo CLOB.

```
LobLib.freeClob(variableClob CLOB inOut)
```

*variableClob*

La variable de tipo CLOB.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“CLOB” en la página 48

“Biblioteca LobLib de EGL” en la página 831

## getBlobLen()

La función de sistema **LobLib.getBlobLen** devuelve el número de bytes en el valor al que hace referencia una variable de tipo BLOB.

```
LobLib.getBlobLen(variableBlob BLOB in)
returns (resultado BIGINT)
```

*resultado*

El número de bytes.

*variableBlob*

La variable de tipo BLOB.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“BLOB” en la página 49

“Biblioteca LobLib de EGL” en la página 831

## getClobLen()

La función de sistema **LobLib.getClobLen** devuelve el número de caracteres a los que hace referencia una variable de tipo CLOB.

```
LobLib.getClobLen(variableClob CLOB in)
returns (resultado BIGINT)
```

*resultado*

El número de caracteres.

*variableBlob*

La variable de tipo CLOB.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“CLOB” en la página 48

“Biblioteca LobLib de EGL” en la página 831

### getStrFromClob()

La función de sistema **LobLib.getStrFromClob** devuelve una serie que corresponde al valor al que hace referencia una variable de tipo CLOB.

```
LobLib.getStrFromClob(variableClob CLOB in)
returns (resultado STRING)
```

*resultado*

La serie devuelta.

*variableClob*

La variable de tipo CLOB.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“CLOB” en la página 48

“Biblioteca LobLib de EGL” en la página 831

### getSubStrFromClob()

La función de sistema **LobLib.getSubStrFromClob** devuelve una subserie del valor al que hace referencia una variable de tipo CLOB.

```
LobLib.getSubStrFromClob(
 variableClob CLOB in,
 pos BIGINT in,
 longitud BIGINT in)
returns (resultado STRING)"
```

*resultado*

Un valor de tipo STRING.

*variableClob*

La variable de tipo CLOB.

*pos*

Identifica la posición numérica del carácter que inicia la subserie. El primer carácter de la variable CLOB está en la posición 1.

*longitud*

Identifica el número de caracteres de la subserie.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“CLOB” en la página 48

“Biblioteca LobLib de EGL” en la página 831

### loadBlobFromFile()

La función de sistema **LobLib.loadBlobFromFile** copia los datos de un archivo especificado en un área de memoria a la que hace referencia una variable de tipo BLOB.

```
LobLib.loadBlobFromFile(
 variableBlob BLOB inOut,
 nombreArchivo STRING in)
```

*variableBlob*

La variable de tipo BLOB.

*nombreArchivo*

El nombre del archivo. El nombre está totalmente calificado o es relativo al directorio desde el que se invoca el programa.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“BLOB” en la página 49

“Biblioteca LobLib de EGL” en la página 831

## loadClobFromFile()

La función de sistema **LobLib.loadClobFromFile** copia los datos de un archivo especificado en un área de memoria a la que hace referencia una variable de tipo CLOB.

```
LobLib.loadClobFromFile(
 variableClob CLOB inOut,
 nombreArchivo STRING in)
```

*variableClob*

La variable de tipo CLOB.

*nombreArchivo*

El nombre del archivo. El nombre está totalmente calificado o es relativo al directorio desde el que se invoca el programa.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“CLOB” en la página 48

“Biblioteca LobLib de EGL” en la página 831

## setClobFromString()

La función de sistema **LobLib.setClobFromString** copia una serie en un área de memoria a la que hace referencia una variable de tipo CLOB.

```
LobLib.setClobFromString(
 variableClob CLOB inOut,
 serie STRING in)
```

*variableClob*

La variable de tipo CLOB.

*serie*

La serie que debe copiarse.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“CLOB” en la página 48

“Biblioteca LobLib de EGL” en la página 831

## setClobFromStringAtPosition()

La función de sistema **LobLib.setClobFromStringAtPosition** copia una serie en el área de memoria a la que hace referencia una variable de tipo CLOB, empezando en una posición especificada del área de memoria.

```
LobLib.setClobFromStringAtPosition(
 variableClob CLOB inOut,
 pos BIGINT in
 serie STRING in)
```

*variableClob*

La variable de tipo CLOB.

*pos*

La posición de carácter en el valor al que hace referencia la *variableClob*. El primer carácter de la variable CLOB está en la posición 1.

*serie*

La serie que debe copiarse.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“CLOB” en la página 48

“Biblioteca LobLib de EGL” en la página 831

## truncateBlob()

La función de sistema **LobLib.truncateBlob** trunca el valor al que hace referencia una variable de tipo BLOB.

```
LobLib.truncateBlob(
 variableBlob BLOB inOut,
 longitud BIGINT in)
```

*variableBlob*

Una variable de tipo BLOB.

*longitud*

El número de bytes de la salida.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“BLOB” en la página 49

“Biblioteca LobLib de EGL” en la página 831

## truncateClob()

La función de sistema **LobLib.truncateClob** trunca el valor al que hace referencia una variable de tipo CLOB.

```
LobLib.truncateClob(
 variableClob CLOB inOut,
 longitud BIGINT in)
```

*variableClob*

Una variable de tipo CLOB.

*longitud*

El número de bytes (no de caracteres) de la salida.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“CLOB” en la página 48

“Biblioteca LobLib de EGL” en la página 831

## updateBlobToFile()

La función de sistema **LobLib.updateBlobToFile** copia los datos a los que hace referencia una variable de tipo BLOB en un archivo especificado. Si el archivo existe, la función primero borra el contenido del archivo; en caso contrario, la función crea el archivo.

```
LobLib.updateBlobToFile(
 variableBlob BLOB inOut,
 nombreArchivo STRING in)
```

*variableBlob*

La variable de tipo BLOB.

*nombreArchivo*

El nombre del archivo. El nombre está totalmente calificado o es relativo al directorio desde el que se invoca el programa.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“BLOB” en la página 49

“Biblioteca LobLib de EGL” en la página 831

## updateClobToFile()

La función de sistema **LobLib.updateClobToFile** copia los datos a los que hace referencia una variable de tipo CLOB en un archivo especificado. Si el archivo existe, la función primero borra el contenido del archivo; en caso contrario, la función crea el archivo.

```
LobLib.updateClobToFile(
 variableClob CLOB inOut,
 nombreArchivo STRING in)
```

*variableClob*

La variable de tipo CLOB.

*nombreArchivo*

El nombre del archivo. El nombre está totalmente calificado o es relativo al directorio desde el que se invoca el programa.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“CLOB” en la página 48

“Biblioteca LobLib de EGL” en la página 831

## Biblioteca MathLib de EGL

La tabla siguiente lista las funciones de la biblioteca del sistema MathLib.

**Nota:** El campo *campoNumérico* es de tipo BIGINT, BIN, DECIMAL, HEX, INT, NUM, NUMC, PACF, SMALLINT, FLOAT o SMALLFLOAT.

Se presupone que un campo de tipo HEX (longitud 8) es un número de coma flotante de 4 bytes de precisión única que es nativo con respecto al entorno de ejecución; se presupone que un campo de tipo HEX (longitud 16) es un número de coma flotante de 8 bytes de precisión doble que es nativo con respecto al entorno de ejecución.

Función de sistema/invocación	Descripción
<i>result</i> = abs ( <i>numericField</i> )	Devuelve el valor absoluto de <i>campoNumérico</i>
<i>result</i> = acos ( <i>numericField</i> )	Devuelve el arco coseno de <i>campoNumérico</i>
<i>result</i> = asin ( <i>numericField</i> )	Devuelve el arco seno de <i>campoNumérico</i>
<i>result</i> = atan ( <i>numericField</i> )	Devuelve el arco tangente de <i>campoNumérico</i>
<i>result</i> = atan2 ( <i>numericField1</i> , <i>numericField2</i> )	Calcula el valor principal del arco tangente de <i>campoNumérico1</i> / <i>campoNumérico2</i> , utilizando los signos de ambos argumentos para determinar el cuadrante del valor de retorno
<i>result</i> = ceiling ( <i>numericField</i> )	Devuelve el entero más pequeño que no sea menor que <i>campoNumérico</i>
<i>result</i> = compareNum ( <i>numericField1</i> , <i>numericField2</i> )	Devuelve un resultado (-1, 0 ó 1) que indica si <i>campoNumérico1</i> es menor, igual o mayor que <i>campoNumérico2</i>
<i>result</i> = cos ( <i>numericField</i> )	Devuelve el coseno de <i>campoNumérico</i>
<i>result</i> = cosh ( <i>numericField</i> )	Devuelve el coseno hiperbólico de <i>campoNumérico</i>
<i>result</i> = exp ( <i>numericField</i> )	Devuelve el valor exponencial de <i>campoNumérico</i>
<i>result</i> = floatingAssign ( <i>numericField</i> )	Devuelve <i>campoNumérico</i> como un número de coma flotante de precisión doble
<i>result</i> = floatingDifference ( <i>numericField1</i> , <i>numericField2</i> )	Devuelve la diferencia entre <i>campoNumérico1</i> y <i>campoNumérico2</i>
<i>result</i> = floatingMod ( <i>numericField1</i> , <i>numericField2</i> )	Calcula el resto de coma flotante de <i>campoNumérico1</i> dividido por <i>campoNumérico2</i> , cuyo resultado tiene el mismo signo que <i>campoNumérico1</i>
<i>result</i> = floatingProduct ( <i>numericField1</i> , <i>numericField2</i> )	Devuelve el producto de <i>campoNumérico1</i> y <i>campoNumérico2</i>
<i>result</i> = floatingQuotient ( <i>numericField1</i> , <i>numericField2</i> )	Devuelve el cociente de <i>campoNumérico1</i> dividido por <i>campoNumérico2</i>
<i>result</i> = floatingSum ( <i>numericField1</i> , <i>numericField2</i> )	Devuelve la suma de <i>campoNumérico1</i> y <i>campoNumérico2</i>

Función de sistema/invocación	Descripción
<i>result = floor (numericField)</i>	Devuelve el entero más grande que no sea mayor que <i>campoNumérico</i>
<i>result = frexp (numericField, integer)</i>	Divide un número en una fracción normalizada en el rango de 0,5 a 1 (que es el valor devuelto) y una potencia de 2 (que se devuelve en <i>entero</i> )
<i>result = Ldexp (numericField, integer)</i>	Devuelve <i>campoNumérico</i> multiplicado por 2 elevado a la potencia de <i>entero</i>
<i>result = log (numericField)</i>	Devuelve el logaritmo natural de <i>campoNumérico</i>
<i>result = log10 (numericField)</i>	Devuelve el logaritmo de base 10 de <i>campoNumérico</i>
<i>result = maximum (numericField1, numericField2)</i>	Devuelve el mayor de <i>campoNumérico1</i> y <i>campoNumérico2</i>
<i>result = minimum (numericField1, numericField2)</i>	Devuelve el menor de <i>campoNumérico1</i> y <i>campoNumérico2</i>
<i>result = modf (numericField1, numericField2)</i>	Divide <i>campoNumérico1</i> en una parte entera y una parte fraccionaria, ambas con el mismo signo que <i>campoNumérico1</i> ; coloca la parte entera en <i>campoNumérico2</i> ; y devuelve la parte fraccionaria
<i>result = pow (numericField1, numericField2)</i>	Devuelve <i>campoNumérico1</i> elevado a la potencia de <i>campoNumérico2</i>
<i>result = precision (numericField)</i>	Devuelve la precisión máxima (en dígitos decimales) para <i>campoNumérico</i>
<i>result = round (numericField[, integer])</i> <i>result = mathLib.round(expresiónNumérica)</i>	Redondea un número o expresión en el valor más cercano (por ejemplo, en el millar más cercano) y devuelve el resultado
<i>result = sin (numericField)</i>	Devuelve el seno de <i>campoNumérico</i>
<i>result = sinh (numericField)</i>	Devuelve el seno hiperbólico de <i>campoNumérico</i>
<i>result = sqrt (numericField)</i>	Devuelve la raíz cuadrada de <i>campoNumérico</i> si <i>campoNumérico</i> es mayor o igual que cero
<i>result = stringAsDecimal (numberAsText)</i>	Acepta un valor de carácter (como por ejemplo "98.6") y devuelve el valor equivalente de tipo DECIMAL
<i>result = stringAsFloat (numberAsText)</i>	Acepta un valor de carácter (como por ejemplo "98.6") y devuelve el valor equivalente de tipo FLOAT
<i>result = stringAsInt (numberAsText)</i>	Acepta un valor de carácter (como por ejemplo "98") y devuelve el valor equivalente de tipo BIGINT
<i>result = tan (numericField)</i>	Devuelve la tangente de <i>campoNumérico</i>
<i>result = tanh (numericField)</i>	Devuelve la tangente hiperbólica de <i>campoNumérico</i>

### **abs()**

La función de sistema **MathLib.abs** devuelve el valor absoluto de un número.

**MathLib.abs**(*campoNumérico* **mathLibNumber** *in*)  
returns (*resultado* **mathLibTypeDependentResult**)

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor absoluto de *elementoNumérico* se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico*

Cualquier elemento numérico o elemento HEX, como se describe en la sección *Matemáticas (palabras del sistema)*.

**MathLib.abs** funciona en cada sistema destino. En relación con los programas Java, EGL utiliza uno de los métodos `abs()` de la clase `Java StrictMath` de modo que el comportamiento en tiempo de ejecución sea el mismo para cada máquina virtual Java.

#### **Ejemplo:**

```
myItem = -5;
result = MathLib.abs(myItem); // resultado = 5
```

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Biblioteca MathLib de EGL” en la página 839

#### **acos()**

La función de sistema **MathLib.acos** devuelve el arco coseno de un argumento, en radianes.

**MathLib.acos**(*campoNumérico* **mathLibNumber** *in*)  
returns (*resultado* **mathLibTypeDependentResult**)

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor devuelto (entre 0,0 y pi) está en radianes y se convierte al formato de *resultado*.

*campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se realice el cálculo. Si el valor no está comprendido entre -1 y 1, se produce un error.

**MathLib.acos** funciona en cada sistema destino. En relación con los programas Java, EGL utiliza el método `acos()` de la clase `Java StrictMath` de modo que el comportamiento en tiempo de ejecución sea el mismo para cada máquina virtual Java.

#### **Ejemplo:**

```
result = MathLib.acos(myItem);
```

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Biblioteca MathLib de EGL” en la página 839

## asin()

La función de sistema **MathLib.asin** devuelve el arco seno de un número que está en el rango de -1 a 1. El resultado es en radianes y está en el rango de  $-\pi/2$  a  $\pi/2$ .

```
MathLib.asin(campoNumérico mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor que devuelve la función **MathLib.asin** se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se llame a la función **mathLib.asin**.

### Ejemplo:

```
result = MathLib.asin(myItem);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## atan()

La función de sistema **MathLib.atan** devuelve el arco tangente de un número. El resultado es en radianes y está en el rango de  $-\pi/2$  y  $\pi/2$ .

```
MathLib.atan(campoNumérico mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor que devuelve **MathLib.atan** se convierte al formato de *resultado*.

*campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se llame a **MathLib.atan**.

### Ejemplo:

```
result = MathLib.atan(myItem);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## atan2()

La función de sistema **MathLib.atan2** calcula el valor principal del arco tangente de  $y/x$ , utilizando los signos de ambos argumentos para determinar el cuadrante del valor de retorno. El resultado es en radianes y está en el rango de  $-\pi$  a  $\pi$ .

```
MathLib.atan2(
 campoNumérico1 mathLibNumber in,
 campoNumérico2 mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor que devuelve **MathLib.atan2** se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico1*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se llame a **MathLib.atan2**. *campoNumérico1* es el valor y.

*campoNumérico2*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se llame a **MathLib.atan2**. *campoNumérico2* es el valor x.

### Ejemplo:

```
myItemY = 1;
myItemX = 5;

// devuelve pi/2
result = MathLib.atan2(myItemY, myItemX);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

### ceiling()

La función de sistema **MathLib.ceiling** devuelve el entero más pequeño que no sea menor que un número especificado.

```
MathLib.ceiling(campoNumérico mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en *Matemáticas (palabras del sistema)*. El entero más pequeño no inferior a *elementoNumérico* se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*.

### Ejemplo:

```
myItem = 4.5;
result = MathLib.ceiling(myItem); // resultado = 5
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## compareNum()

La función de sistema **MathLib.compareNum** devuelve un resultado (-1, 0 ó 1) que indica si el primero de dos números es menor, igual o mayor que el segundo.

```
MathLib.compareNum(
 campoNumérico1 mathLibNumber in,
 campoNumérico2 mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales. Este elemento recibe uno de los siguientes valores:

- 1 *campoNumérico1* es menor que *campoNumérico2*.
- 0 *campoNumérico1* es igual que *campoNumérico2*.
- 1 *campoNumérico1* es mayor que *campoNumérico2*.

*campoNumérico1*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*.

*campoNumérico2*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*.

### Ejemplo:

```
myItem01 = 4
myItem02 = 7

result = MathLib.compareNum(myItem01,myItem02);

// resultado = -1
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## cos()

La función de sistema **MathLib.cos** devuelve el coseno de un número. El valor devuelto está en el rango de -1 a 1.

```
MathLib.cos(campoNumérico mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor que devuelve **MathLib.cos** se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se llame a **MathLib.cos**.

### Ejemplo:

```
result = MathLib.cos(myItem);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## cosh()

La función de sistema **MathLib.cosh** devuelve el coseno hiperbólico de un número.

```
MathLib.cosh(campoNumérico mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor que devuelve **mathLib.cosh** se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se llame a **mathLib.cosh**.

### Ejemplo:

```
result = MathLib.cosh(myItem);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## exp()

La función de sistema **MathLib.exp** devuelve  $e$  elevado a la potencia de un número.

```
MathLib.exp(campoNumérico mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en *MathLib*. El valor que devuelve **MathLib.exp** se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico*

Cualquier elemento numérico o HEX, como se describe en *MathLib*. El elemento se convierte a coma flotante de precisión doble antes de que se llame a **MathLib.exp**.

### Ejemplo:

```
result = MathLib.exp(myItem);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## floatingAssign()

La función de sistema **MathLib.floatingAssign** devuelve *elementoNumérico* como un número de coma flotante de precisión doble. La función asigna el valor de los

elementos BIN, DECIMAL, NUM, NUMC o PACKF a números de coma flotante que están definidos como elementos HEX, y viceversa.

```
MathLib.floatingAssign(campoNumérico mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El número de coma flotante se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se asigne al resultado.

#### **Ejemplo:**

```
result = MathLib.floatingAssign(myItem);
```

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Biblioteca MathLib de EGL” en la página 839

### **floatingDifference()**

La función de sistema **MathLib.floatingDifference** resta el segundo de dos números del primero y devuelve la diferencia. La función se implementa utilizando aritmética de coma flotante de precisión doble.

```
MathLib.floatingDifference(
 campoNumérico1 mathLibNumber in,
 campoNumérico2 mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. La diferencia se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico1*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule la diferencia.

*campoNumérico2*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule la diferencia.

#### **Ejemplo:**

```
result = MathLib.floatingDifference(myItem01,myItem02);
```

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Biblioteca MathLib de EGL” en la página 839

## floatingMod()

La función de sistema **MathLib.floatingMod** devuelve el resto de coma flotante de un número dividido por otro. El resultado tiene el mismo signo que el numerador. Se lanza una excepción de dominio si el denominador es igual a cero.

```
MathLib.floatingMod(
 campoNumérico1 mathLibNumber in,
 campoNumérico2 mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El resto de coma flotante se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico1*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule *resultado*.

*campoNumérico2*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule *resultado*.

### Ejemplo:

```
result = MathLib.floatingMod(myItem01,myItem02);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## floatingProduct()

La función de sistema **MathLib.floatingProduct** devuelve el producto de dos números. La función se implementa utilizando aritmética de coma flotante de precisión doble.

```
MathLib.floatingProduct(
 campoNumérico1 mathLibNumber in,
 campoNumérico2 mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El producto se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico1*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule *resultado*.

*campoNumérico2*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule *resultado*.

### Ejemplo:

```
result = MathLib.floatingProduct(myItem01,myItem02);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## floatingQuotient()

La función de sistema **MathLib.floatingQuotient** devuelve el cociente de un número dividido por otro. Se lanza una excepción de dominio si el denominador es igual a cero. La función se implementa utilizando aritmética de coma flotante de precisión doble.

```
MathLib.floatingQuotient(
 campoNumérico1 mathLibNumber in,
 campoNumérico2 mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El cociente se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico1*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule el cociente.

*campoNumérico2*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule el cociente.

### Ejemplo:

```
result = MathLib.floatingQuotient(myItem01,myItem02);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## floatingSum()

La función de sistema **MathLib.floatingSum** devuelve la suma de dos números. La función se implementa utilizando aritmética de coma flotante de precisión doble.

```
MathLib.floatingSum(
 campoNumérico1 mathLibNumber in,
 campoNumérico2 mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. La suma se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico1*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule la suma.

*campoNumérico2*

Cualquier elemento numérico o HEX, como se describe en la sección

*Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule la suma.

**Ejemplo:**

```
result = MathLib.floatingSum(myItem01,myItem02);
```

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca MathLib de EGL” en la página 839

**floor()**

La función de sistema **MathLib.floor** devuelve el entero más alto que no sea mayor que un número especificado.

```
MathLib.floor(campoNumérico mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El entero más grande no superior a *campoNumérico* se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*.

**Ejemplo:**

```
myItem = 4.6;
result = MathLib.floor(myItem); // resultado = 4
```

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca MathLib de EGL” en la página 839

**frexp()**

La función de sistema **MathLib.frexp** divide un número en una fracción normalizada en el rango de 0,5 a 1 (que se devuelve como *resultado*) y una potencia de 2 (que se devuelve en *exponente*).

```
MathLib.frexp(
 campoNumérico mathLibNumber in,
 exponente mathLibInteger inOut)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. La fracción de coma flotante se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule *resultado*.

*exponente*

Elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

**Ejemplo:**

```
result = MathLib.frexp(myItem,myInteger);
```

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca MathLib de EGL” en la página 839

**Ldexp()**

La función de sistema **MathLib.Ldexp** devuelve el valor de un número especificado que se multiplica por el siguiente valor: dos elevado a la potencia de *exponente*.

```
MathLib.Ldexp(
 campoNumérico mathLibNumber in,
 exponente mathLibInteger in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor calculado se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule *resultado*.

*exponente*

Elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

**Ejemplo:**

```
result = MathLib.Ldexp(myItem,myInteger);
```

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca MathLib de EGL” en la página 839

**log()**

La función de sistema **MathLib.log** devuelve el logaritmo natural de un número.

```
MathLib.log(campoNumérico mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor que devuelve la función *mathLib.log* se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule *resultado*.

**Ejemplo:**

```
result = MathLib.log(myItem);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## log10()

La función de sistema **MathLib.log10** devuelve el logaritmo en base 10 de un número.

```
MathLib.log10(campoNumérico mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

### resultado

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor que devuelve la función log10 se convierte al formato de *resultado* y se devuelve en *resultado*.

### campoNumérico

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule *resultado*.

### Ejemplo:

```
result = MathLib.log10(myItem);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## maximum()

La función de sistema **MathLib.maximum** devuelve el mayor de dos números.

```
MathLib.maximum(
 campoNumérico1 mathLibNumber in,
 campoNumérico2 mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

### resultado

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El mayor de dos números se convierte al formato de *resultado* y se devuelve en *resultado*.

### campoNumérico1

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*.

### campoNumérico2

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*.

### Ejemplo:

```
result = MathLib.maximum(myItem01,myItem02);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## minimum()

La función de sistema **MathLib.minimum** devuelve el menor de dos números.

```
MathLib.minimum(
 campoNumérico1 mathLibNumber in,
 campoNumérico2 mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El menor de dos números se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico1*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*.

*campoNumérico2*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*.

### Ejemplo:

```
result = MathLib.minimum(myItem01,myItem02);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## modf()

La función de sistema **MathLib.modf** divide un número en una parte integral y en una fraccionaria, ambas con el mismo signo que el número. La parte fraccionaria se devuelve en *resultado* y la parte integral se devuelve en *campoNumérico2*.

```
MathLib.modf(
 campoNumérico1 mathLibNumber in,
 campoNumérico2 mathLibNumber inOut)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. La parte fraccionaria de *campoNumérico1* se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico1*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*.

*campoNumérico2*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. La parte integral de *campoNumérico1* se convierte al formato de *campoNumérico2* y se devuelve en *campoNumérico2*.

### Ejemplo:

```
result = MathLib.modf(myItem01,myItem02);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

### pow()

La función de sistema **MathLib.pow** devuelve un número elevado a la potencia de un segundo número. Se lanza una excepción de dominio si en `pow(x,y)` el valor de `x` es negativo e `y` no es un valor entero, o si el valor de `x` es 0,0 e `y` es un valor negativo.

```
MathLib.pow(
 campoNumérico1 mathLibNumber in,
 campoNumérico2 mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

#### *resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El resultado de `mathLib.pow` se convierte al formato de *resultado* y se devuelve en *resultado*.

#### *campoNumérico1*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule *resultado*.

#### *campoNumérico2*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule *resultado*.

### Ejemplo:

```
result = MathLib.pow(myItem01,myItem02);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

### precision()

La función de sistema **MathLib.precision** devuelve la precisión máxima (en dígitos decimales) de un número. Para números de coma flotante (HEX de 8 dígitos para número de coma flotante de precisión estándar o HEX de 16 dígitos para número de coma flotante de precisión doble), la precisión es el número máximo de dígitos decimales que pueden representarse en el número para el sistema en el que se ejecuta el programa.

```
MathLib.precision(campoNumérico mathLibNumber in)
returns (resultado INT)
```

#### *resultado*

Un elemento que recibe la precisión de *elementoNumérico*. El elemento *resultado* se define como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

#### *campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*.

### Ejemplo:

```
result = MathLib.precision(myItem);
```

## Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

## Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## round()

La función de sistema **MathLib.round** redondea un número o expresión en el valor más cercano (por ejemplo, en el millar más cercano) y devuelve el resultado.

```
MathLib.round(
 campoNumérico mathLibNumber in
 [, potenciaDe10 mathLibInteger in
]
)
returns (resultado mathLibTypeDependentResult)
MathLib.round(expresiónNumérica anyNumericExpression in
returns (resultado mathLibTypeDependentResult)
```

### *resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor generado por la operación de redondeo se convierte al formato de *resultado* y se devuelve en *resultado*.

La longitud máxima soportada en este caso es de 31 en lugar de 32, ya que el redondeo se produce del siguiente modo:

- Se añade cinco al dígito de *resultado* con una precisión superior en uno a la del dígito de resultado
- Se trunca el resultado

Si se utilizan más de 31 dígitos en el cálculo y EGL no puede determinar la violación durante el desarrollo, se produce un desbordamiento numérico durante la ejecución.

### *campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*.

### *expresiónNumérica*

Una expresión numérica que no sea simplemente un elemento numérico. Si especifica un operador, no puede especificar un valor para *entero*.

No puede utilizar **MathLib.round** con el operador de resto (%).

### *potenciaDe10*

Un entero que determina el valor al que se redondea el número:

- Si el entero es positivo, el número se redondea en el valor más cercano igual a 10 elevado a la potencia de *potenciaDe10*. Por ejemplo, si el entero es 3, el número se redondea en el millar más cercano.
- La misma operación se realiza si el entero es cero o negativo; en ese caso, el número se redondea en el número de posiciones decimales especificado.

Si no especifica *potenciaDe10*, **MathLib.round** se redondea en el número de posiciones decimales de *resultado*.

El entero se define como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

**Ejemplos:** En el siguiente ejemplo, el elemento balance se redondea al millar más próximo:

```
balance = 12345.6789;
rounder = 3;
balance = MathLib.round(balance, rounder);
// El valor de balance ahora es 12000.0000
```

En el siguiente ejemplo, se utiliza un valor `rounder` de -2 para redondear `balance` a dos posiciones decimales:

```
balance = 12345.6789;
rounder = -2;
balance = mathLib.round(balance, rounder);
// El valor de balance ahora es 12345.6800
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## sin()

La función de sistema **MathLib.sin** que devuelve el seno de un número. El resultado está en el rango de -1 a 1.

```
MathLib.sin(campoNumérico mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor que devuelve la función **MathLib.sin** se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule *resultado*.

### Ejemplo:

```
result = MathLib.sin(myItem);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## sinh()

La función de sistema **MathLib.sinh** devuelve el seno hiperbólico de un número.

```
MathLib.sinh(campoNumérico mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

*resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor que devuelve la función **MathLib.sinh** se convierte al formato de *resultado* y se devuelve en *resultado*.

*campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule *resultado*.

### Ejemplo:

```
result = MathLib.sinh(myItem);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## sqrt()

La función matemática **MathLib.sqrt** devuelve la raíz cuadrada de un número. La función actúa sobre cualquier número que sea mayor o igual que cero.

```
MathLib.sqrt(campoNumérico mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

### *resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor que devuelve la función **MathLib.sqrt** se convierte al formato de *resultado* y se devuelve en *resultado*.

### *campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule *resultado*.

### Ejemplo:

```
result = MathLib.sqrt(myItem);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## stringAsDecimal()

La función de sistema **MathLib.stringAsDecimal** acepta un valor de carácter (como por ejemplo "98.6") y devuelve el valor equivalente de tipo DECIMAL.

```
MathLib.stringAsDecimal(númeroComoTexto STRING in)
returns (resultado DECIMAL)
```

### *resultado*

Un valor de tipo DECIMAL. El campo receptor puede tener cualquier posición decimal y cualquier longitud.

EGL permite un total de 32 dígitos a ambos lados de la coma decimal. La coma decimal (si existe) es específica del entorno local Java.

Para obtener información detallada sobre las implicaciones de asignar valores a campos de diferentes tipos, consulte el apartado *Asignaciones*.

### *númeroComoTexto*

Un campo de caracteres o serie de literal, que puede incluir un carácter de signo inicial.

### Ejemplo:

```
myField = "-5.243";

// resultado = -5.243
result = MathLib.stringAsDecimal(myField);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Asignaciones” en la página 374

“Biblioteca MathLib de EGL” en la página 839

### stringAsFloat()

La función de sistema **MathLib.stringAsFloat** acepta un valor de carácter (como por ejemplo "98.6") y devuelve el valor equivalente de tipo FLOAT.

```
MathLib.stringAsFloat(númeroComoTexto STRING in)
returns (resultado FLOAT)
```

#### *resultado*

Un valor de tipo FLOAT. El campo receptor puede tener cualquier posición decimal y cualquier longitud. La coma decimal (si existe) es específica del entorno local Java.

Para obtener información detallada sobre las implicaciones de asignar valores a campos de diferentes tipos, consulte el apartado *Asignaciones*.

#### *númeroComoTexto*

Un campo de caracteres o serie de literal, que puede incluir un carácter de signo inicial.

### Ejemplo:

```
myField = "-5.243";

// resultado = -5.243
result = MathLib.stringAsFloat(myField);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Asignaciones” en la página 374

“Biblioteca MathLib de EGL” en la página 839

### stringAsInt()

La función de sistema **MathLib.stringAsInt** acepta un valor de carácter (como por ejemplo "98") y devuelve el valor equivalente de tipo BIGINT.

```
MathLib.stringAsInt(númeroComoTexto STRING in)
returns (resultado BIGINT)
```

#### *resultado*

Un valor de tipo BIGINT.

Para obtener información detallada sobre las implicaciones de asignar valores a campos de diferentes tipos, consulte el apartado *Asignaciones*.

#### *númeroComoTexto*

Un campo de caracteres o serie de literal, que puede incluir un carácter de signo inicial.

### Ejemplo:

```
myField = "-5";

// resultado = -5
result = MathLib.stringAsInt(myField);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## tan()

La función de sistema **MathLib.tan** devuelve la tangente de un número.

```
MathLib.tan(campoNumérico mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

### *resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor que devuelve la función **MathLib.tan** se convierte al formato de *resultado* y se devuelve en *resultado*.

### *campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule *resultado*.

### Ejemplo:

```
result = MathLib.tan(myItem);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## tanh()

La función de sistema **MathLib.tanh** devuelve la tangente hiperbólica de un número. El resultado está en el rango de -1 a 1.

```
MathLib.tanh(campoNumérico mathLibNumber in)
returns (resultado mathLibTypeDependentResult)
```

### *resultado*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El valor que devuelve la función **MathLib.tanh** se convierte al formato de *resultado* y se devuelve en *resultado*.

### *campoNumérico*

Cualquier elemento numérico o HEX, como se describe en la sección *Matemáticas (palabras del sistema)*. El elemento se convierte a coma flotante de precisión doble antes de que se calcule *resultado*.

### Ejemplo:

```
result = MathLib.tanh(myItem);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca MathLib de EGL” en la página 839

## recordName.resourceAssociation

Cuando el programa realiza una operación de E/S en un registro, la E/S se realiza en el archivo físico cuyo nombre se encuentra en la variable específica del registro *nombreRegistro*. **resourceAssociation**. La variable se inicializa de acuerdo con el componente **resourceAssociation** utilizado durante la generación; para obtener detalles, consulte el apartado *Asociaciones de recursos y tipos de archivos*. Puede cambiar el nombre de recurso del sistema durante la ejecución colocando un valor distinto en **resourceAssociation**.

En la mayoría de los casos, debe utilizar la sintaxis *nombreRegistro.resourceAssociation*. Sin embargo, no es necesario especificar un nombre de recurso si EGL puede determinar el registro que se pretende utilizar, como ocurre en cada uno de los casos siguientes:

- La E/S sólo se realiza en un registro del programa
- **resourceAssociation** se utiliza en una función que realiza la E/S en sólo un registro
- La E/S se realiza en varios registros del programa, pero todos los registros tienen el mismo nombre de archivo; en este caso, el primer registro que aparece como objeto de E/S se utiliza como calificador implícito.

Puede utilizar **resourceAssociation** de las siguientes maneras:

- Como operando origen o destino de una sentencia assignment
- Como elemento de una expresión lógica en una sentencia **case**, **if** o **while**
- Como argumento de una sentencia **return** o **exit**

Las características de **resourceAssociation** son las siguientes:

### Tipo primitivo

CHAR

### Longitud de datos

Varía en función del tipo de archivo

### ¿Se guarda a lo largo de los segmentos?

Sí

### Consideraciones de definición

El valor trasladado a *nombreRegistro*. **resourceAssociation** debe ser un nombre de recurso del sistema válido para el sistema y para el tipo de archivo especificados al generar el programa. Si más de un registro especifica el mismo nombre de archivo, la modificación de **resourceAssociation** para cualquier registro con dicho nombre de archivo cambia el valor de **resourceAssociation** para todos los registros del programa con el mismo nombre de archivo.

Si un recurso del sistema identificado en el valor de **resourceAssociation** está abierto cuando se modifica la variable específica del registro, el recurso del sistema que *estaba* en dicha variable se cierra en la siguiente circunstancia: una opción de E/S se ejecuta en un registro que tiene el mismo nombre de archivo EGL que el registro que califica a **resourceAssociation**.

Si dos programas utilizan el mismo nombre de archivo, cada una de las variables **resourceAssociation** específicas del registro deben contener el mismo valor. En caso contrario, el recurso del sistema abierto previamente se cierra cuando se abre uno nuevo.

El resultado de una comparación de **resourceAssociation** con otro valor es true sólo si la coincidencia es exacta. Si, por ejemplo, inicializa **resourceAssociation** con un valor en minúsculas, este valor sólo coincide con un valor en minúsculas.

**Archivos compartidos entre programas:** Puede establecer el nombre del recurso del sistema durante la generación o durante la ejecución:

#### **Durante la generación**

Si dos programas de la misma unidad de ejecución acceden al mismo archivo lógico, debe especificar el mismo nombre de recurso del sistema durante la generación para asegurarse de que ambos programas accedan al mismo archivo físico durante la ejecución.

#### **Durante la ejecución**

Si utiliza *nombreRegistro*. **resourceAssociation**, cada programa que acceda al archivo debe establecer **resourceAssociation** para el archivo. Si dos programas de la misma unidad de ejecución acceden al mismo archivo lógico, cada programa debe establecer **resourceAssociation** en el mismo nombre de recurso del sistema para asegurar que ambos programas accedan al mismo archivo físico durante la ejecución.

Si varios programas comparten un recurso del sistema, cada programa que accede al recurso debe establecer **resourceAssociation** para hacer referencia al mismo recurso. Además, si dos programas de la misma unidad de ejecución acceden al mismo archivo lógico, cada programa debe establecer **resourceAssociation** en el mismo nombre de recurso del sistema durante la generación para asegurar que ambos programas accedan al mismo recurso del sistema durante la ejecución.

**Registros MQ:** El nombre de recurso del sistema para registros MQ define el nombre de gestor de colas y el nombre de cola. Especifique el nombre en el siguiente formato:

*nombreGestorColas:nombreCola*

*nombreGestorColas*

Nombre del gestor de colas.

*nombreCola*

Nombre de la cola.

Como se muestra, los nombres están separados por un carácter de dos puntos. Sin embargo, *nombreGestorColas* y el carácter de dos puntos pueden omitirse. El nombre de recurso del sistema se utiliza como valor inicial para el elemento **resourceAssociation** específico del registro e identifica la cola por omisión asociada al registro. Para obtener más detalles, consulte el apartado *Soporte de MQSeries*.

### **Ejemplo**

```
if (process == 1)
 myrec.resourceAssociation = "myFile.txt";
else
 myrec.resourceAssociation = "myFile02.txt";
end
```

#### **Conceptos relacionados**

“Soporte de MQSeries” en la página 265

“Asociaciones de recursos y tipos de archivo” en la página 304

#### **Consulta relacionada**

## Biblioteca ReportLib de EGL

*ReportLib*, la biblioteca de informes de EGL es una biblioteca del sistema que establece una infraestructura que contiene todos los componentes necesarios para interactuar con la biblioteca JasperReports. La biblioteca de informes EGL incluye los componentes siguientes:

- Funciones, variables y constantes que se utilizan para los siguientes fines:
  - Interactuar con funciones de biblioteca JasperReports
  - Definir, establecer y recuperar el origen de datos de un informe
  - Exportar un informe cumplimentado a distintos formatos de archivo
  - Manipular el contenido del informe y procesar los datos de informe
- Registros que contienen nombres de archivos que almacenan el diseño del informe, el informe cumplimentado y el informe exportado.
- El informe

La biblioteca de informes incluye las funciones siguientes:

Función de sistema/invocación	Descripción
<code>addReportParameter(informe, serieParámetro, valorParámetro)</code>	Cumplimenta el informe que utiliza el origen de datos especificado.
<code>fillReport(informe, origen)</code>	Exporta el informe cumplimentado en el formato especificado.
<code>exportReport(informe, formato)</code>	Añade un valor a la lista de parámetros del informe.
<code>resetReportParameters(informe)</code>	Elimina todos los parámetros utilizados para un informe determinado.

Las siguientes funciones sólo se invocan dentro de manejadores de informes:

Función de sistema/invocación	Descripción
<code>addReportData(rd, nombreConjuntoDatos)</code>	Añade el objeto de datos de informe con el nombre especificado al manejador de informes actual.
<code>result = getReportData(dataSetName)</code>	Recupera el registro de datos de informe con el nombre especificado. El valor devuelto es de tipo <code>ReportData</code> .
<code>result = getReportParameter(parameter)</code>	Devuelve el valor del parámetro especificado del informe que se está cumplimentando.
<code>result = getFieldValue(fieldName)</code>	Devuelve el valor del campo especificado para la fila que se está procesando actualmente. El valor devuelto es de tipo ANY.
<code>result = getReportVariableValue(variable)</code>	Devuelve el valor de la variable especificada del informe que se está cumplimentando. El valor devuelto es de tipo ANY.
<code>setReportVariableValue(variable, valor)</code>	Establece el valor de la variable especificada en el valor proporcionado.

**Nota:** Si suprime un informe de EGL, deberá eliminar todas las referencias al informe.

#### Conceptos relacionados

“Orígenes de datos” en la página 212

“Visión general del proceso de creación de informes de EGL” en la página 210

“Visión general de los informes de EGL” en la página 209

### addReportData()

La función del sistema **ReportLib.addReportData** añade el objeto de datos de informe con el nombre especificado al manejador de informes actual.

```
ReportLib.addReportData(
 rd ReportData in,
 nombreConjuntoDatos STRING in)
```

*rd*

*nombreConjuntoDatos*

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“Visión general de los informes de EGL” en la página 209

“Visión general del proceso de creación de informes de EGL” en la página 210

#### Consulta relacionada

“Biblioteca ReportLib de EGL” en la página 861

“addReportParameter()”

“fillReport()” en la página 863

“exportReport()” en la página 863

### addReportParameter()

El diagrama de sintaxis de la función **ReportLib.addReportParameter** es el siguiente:

```
ReportLib.addReportParameter(
 informe Report in,
 serieParámetro STRING in,
 valorParámetro any in)
```

*informe*

El nombre del informe.

*serieParámetro*

Un parámetro que desea utilizar en el informe.

*valorParámetro*

El valor del parámetro especificado en *serieParámetro*.

Antes de cumplimentar un informe, EGL puede pasar un conjunto de parámetros que establecen valores a utilizar en el informe o que alteran temporalmente parámetros especificados en el diseño de informes XML. La función **ReportLib.addReportParameter** añade el valor del parámetro especificado a la lista de parámetros del informe.

**Nota:** Consulte la documentación de JasperReports para obtener información acerca de los tipos de datos y los parámetros de JasperReports.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754  
Visión general del informe EGL  
Visión general del proceso de creación de informes de EGL

### Consulta relacionada

Biblioteca de informes de EGL  
Función `ReportLib.fillReport`  
Función `ReportLib.exportReport`  
Función `ReportLib.resetReportParameters`

## exportReport()

La función de sistema **ReportLib.exportReport** exporta el informe cumplimentado en el formato especificado.

El diagrama siguiente ilustra la sintaxis de dicha función:

```
ReportLib.exportReport(
 informe Report in,
 formato ExportFormat in)
```

*informe*

El informe que se exporta.

*formato*

El formato y la extensión de archivo para el informe exportado.

Los valores son los de la enumeración **ExportFormat**:

**csv**

La salida muestra un valor separado del siguiente mediante una coma; **csv** indica valores separados por comas.

**html**

La salida está en formato HTML.

**pdf**

La salida está en formato Adobe Acrobat PDF.

**text**

La salida está en formato de texto ASCII.

### Conceptos relacionados

“Visión general de los informes de EGL” en la página 209  
“Visión general del proceso de creación de informes de EGL” en la página 210  
“Enumeraciones en EGL” en la página 484  
“Diagrama de sintaxis para funciones EGL” en la página 754

### Tareas relacionadas

“Exportar informes” en la página 228

### Consulta relacionada

“addReportParameter()” en la página 862  
“Biblioteca ReportLib de EGL” en la página 861  
“fillReport()”  
“resetReportParameters()” en la página 866

## fillReport()

El diagrama de sintaxis de la función **ReportLib.fillReport** es el siguiente:

```
ReportLib.fillReport(
 informe Report in,
 origen DataSource in)
```

*informe*

El informe que debe rellenarse con datos.

*origen*

El origen de los datos que se utilizan para rellenar el informe.

Considere este ejemplo, que muestra cómo se asocia una variable de tipo **reportData** con el informe:

```
eglReport Report;
eglReportData ReportData;
eglReport.reportData = eglReportData;
```

El *origen* indica qué campo debe utilizarse en la variable de tipo **ReportData**. Cada valor de *origen* no es un nombre de campo, sino un valor de la enumeración **DataSource**:

#### **databaseConnection**

Utilice la variable a la que se hace referencia en el campo **connectionName** de la variable **reportData**, como en este ejemplo:

```
eglReportData.connectionName = "mycon";
```

En este caso, la sentencia SQL que accede a los datos se encuentra en el archivo de diseño del informe, que se crea fuera de EGL.

#### **reportData**

Utilice la variable a la que se hace referencia en el campo **data** de la variable **reportData**, como en este ejemplo:

```
// matriz de registros con
datos
myRecords customerRecord[];

eglReportData.data = myRecords;
```

#### **sqlStatement**

Utilice la sentencia SQL identificada en el campo **sqlStatement** de la variable **reportData**, como en este ejemplo:

```
mySQLString = "Select * From MyTable";
eglReportData.sqlStatement = mySQLString;
```

A continuación figura un ejemplo de invocación:

```
ReportLib.fillReport (eglReport, DataSource.sqlStatement);
```

#### **Conceptos relacionados**

“Visión general de los informes de EGL” en la página 209

“Visión general del proceso de creación de informes de EGL” en la página 210

“Enumeraciones en EGL” en la página 484

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Orígenes de datos” en la página 212

“Biblioteca ReportLib de EGL” en la página 861

“addReportParameter()” en la página 862

“exportReport()” en la página 863

“resetReportParameters()” en la página 866

## getFieldValue()

La función **ReportLib.getFieldValue** devuelve el valor del campo especificado para la fila que se está procesando actualmente.

```
ReportLib.getFieldValue(nombreCampo STRING in)
returns (resultado ANY)
```

*resultado*

*nombreCampo*

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“Visión general de los informes de EGL” en la página 209

“Visión general del proceso de creación de informes de EGL” en la página 210

### Consulta relacionada

“addReportParameter()” en la página 862

“fillReport()” en la página 863

“Biblioteca ReportLib de EGL” en la página 861

“exportReport()” en la página 863

## getReportData()

La función del sistema **ReportLib.getReportData** recupera el registro de datos de informe con el nombre especificado.

```
ReportLib.getReportData(nombreConjuntoDatos STRING in)
returns (resultado ReportData)
```

*resultado*

*nombreConjuntoDatos*

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“Visión general de los informes de EGL” en la página 209

“Visión general del proceso de creación de informes de EGL” en la página 210

### Consulta relacionada

“addReportParameter()” en la página 862

“Biblioteca ReportLib de EGL” en la página 861

“exportReport()” en la página 863

“fillReport()” en la página 863

## getReportParameter()

La función **ReportLib.getReportParameter** devuelve el valor del parámetro especificado del informe que se está cumplimentando.

```
ReportLib.getReportParameter(parámetro STRING in)
returns (resultado ANY)
```

*resultado*

*parámetro*

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“Visión general de los informes de EGL” en la página 209

“Visión general del proceso de creación de informes de EGL” en la página 210

### Consulta relacionada

“Biblioteca ReportLib de EGL” en la página 861  
“addReportParameter()” en la página 862  
“fillReport()” en la página 863  
“exportReport()” en la página 863

## getReportVariableValue()

La función del sistema **ReportLib.getReportVariableValue** devuelve el valor de la variable especificada del informe que se está cumplimentando.

```
ReportLib.getReportVariableValue(variable STRING in)
returns (resultado ANY)
```

*resultado*

*variable*

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754  
“Visión general de los informes de EGL” en la página 209  
“Visión general del proceso de creación de informes de EGL” en la página 210

### Consulta relacionada

“addReportParameter()” en la página 862  
“Biblioteca ReportLib de EGL” en la página 861  
“exportReport()” en la página 863  
“fillReport()” en la página 863

## resetReportParameters()

El diagrama de sintaxis de la función **ReportLib.resetReportParameters** es el siguiente:

```
ReportLib.resetReportParameters(informe Report in)
```

*informe*

El nombre del informe que contiene los parámetros que desea eliminar.

La función **ReportLib.resetReportParameters** elimina todos los parámetros de EGL utilizados para un informe determinado.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754  
“Visión general de los informes de EGL” en la página 209  
“Visión general del proceso de creación de informes de EGL” en la página 210

### Consulta relacionada

“Biblioteca ReportLib de EGL” en la página 861  
“addReportParameter()” en la página 862  
“exportReport()” en la página 863  
“fillReport()” en la página 863

## setReportVariableValue()

La función **ReportLib.setReportVariableValue** establece el valor de la variable especificada en un valor proporcionado a la función.

```
ReportLib.setReportVariableValue(
 variable STRING in,
 valor Any in)
```

*variable*

valor

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“Visión general del proceso de creación de informes de EGL” en la página 210

“Visión general de los informes de EGL” en la página 209

### Consulta relacionada

“addReportParameter()” en la página 862

“Biblioteca ReportLib de EGL” en la página 861

“fillReport()” en la página 863

“exportReport()” en la página 863

## Biblioteca StrLib de EGL

La tabla que sigue muestra las funciones de sistema de la biblioteca **StrLib** y va seguida de tablas que muestran las variables y constantes de dicha biblioteca.

Función de sistema e invocación	Descripción
<i>result</i> = characterAsInt ( <i>text</i> )	Convierte una serie de caracteres en una serie de enteros correspondiente al primer carácter de la expresión de caracteres.
<i>result</i> = clip ( <i>text</i> )	Suprime espacios en blanco finales y nullos del final de las series de caracteres devueltas.
<i>result</i> = compareStr ( <i>target</i> , <i>targetSubstringIndex</i> , <i>targetSubstringLength</i> , <i>source</i> , <i>sourceSubstringIndex</i> , <i>sourceSubstringLength</i> )	Compara dos subseries de acuerdo con su orden ASCII o EBCDIC durante la ejecución y devuelve un valor (-1, 0 o 1) para indicar cuál es mayor.
<i>result</i> = concatenate ( <i>target</i> , <i>source</i> )	Concatena <i>destino</i> y <i>origen</i> ; coloca la nueva serie en <i>destino</i> ; y devuelve un entero que indica si <i>destino</i> era suficientemente largo para contener la nueva serie
<i>result</i> = concatenateWithSeparator ( <i>target</i> , <i>source</i> , <i>separator</i> )	Concatena <i>destino</i> y <i>origen</i> , insertando <i>separador</i> entre ellos; coloca la nueva serie en <i>destino</i> ; y devuelve un entero que indica si <i>destino</i> era suficientemente largo para contener la nueva serie
copyStr ( <i>destino</i> , <i>índiceSubserieDestino</i> , <i>longitudSubserieDestino</i> , <i>origen</i> , <i>índiceSubserieOrigen</i> , <i>longitudSubserieOrigen</i> )	Copia una subserie en otra
<i>result</i> = findStr ( <i>source</i> , <i>sourceSubstringIndex</i> , <i>sourceSubstringLength</i> , <i>searchString</i> )	Busca la primera aparición de una subserie dentro de una serie.
<i>result</i> = formatDate ( <i>valorFecha</i> [, <i>formatoFecha</i> ])	Da formato a un valor de fecha y devuelve un valor de tipo STRING. El formato predeterminado es el formato especificado en el entorno local actual.
<i>result</i> = formatNumber ( <i>expresiónNumérica</i> , <i>formatoNumérico</i> )	Devuelve un número como serie con formato.
<i>result</i> = formatTime ( <i>valorHora</i> [, <i>formatoHora</i> ])	Da formato a un parámetro en un valor de hora y devuelve un valor de tipo STRING. El formato predeterminado es el formato especificado en el entorno local actual.

Función de sistema e invocación	Descripción
<code>result = formatTimeStamp (valorIndicaciónHora [, formatoIndicaciónHora])</code>	Da formato a un parámetro en un valor de indicación de la hora y devuelve un valor de tipo STRING. El formato DB2 es el formato por omisión.
<code>result = getNextToken (target, source, sourceSubstringIndex, sourceStringLength, characterDelimiter)</code>	Busca en una serie el siguiente símbolo y lo copia en <i>destino</i>
<code>result = integerAsChar (integer)</code>	Convierte una serie de entero en una serie de caracteres.
<code>result = lowerCase (text)</code>	Convierte todos los valores en mayúsculas de una serie de caracteres en valores en minúsculas. Los valores numéricos y en minúsculas existentes no se ven afectados.
<code>setBlankTerminator (destino)</code>	Sustituye un terminador nulo y los caracteres subsiguientes (si los hay) de una serie por espacios, de modo que un valor de serie devuelto desde un programa C o C++ pueda funcionar correctamente en un programa generado por EGL
<code>setNullTerminator (destino)</code>	Cambia todos los espacios finales de una serie por nulos
<code>setSubStr (destino, índiceSubserieDestino, longitudSubserieDestino, origen)</code>	Sustituye cada carácter de una subserie por un carácter especificado
<code>result =spaces (characterCount)</code>	Devuelve una serie de una longitud especificada.
<code>result = strLen (source)</code>	Devuelve el número de bytes de un elemento, excluyendo los espacios finales o los nulos
<code>result = textLen (source)</code>	Devuelve el número de bytes de una expresión de texto, excluyendo los espacios finales o los nulos
<code>result = upperCase (characterItem)</code>	Convierte todos los valores en minúsculas de una serie de caracteres en valores en mayúsculas. Los valores numéricos y en mayúsculas existentes no se ven afectados.

La tabla siguiente muestra las variables de sistema de la biblioteca **StrLib**.

Variable de sistema	Descripción
<code>defaultDateFormat</code>	Especifica el valor de <b>defaultDateFormat</b> , que es una de las diversas máscaras que pueden utilizarse para crear la serie devuelta por la función <b>StrLib.formatDate</b> .
<code>defaultMoneyFormat</code>	Especifica el valor de <b>defaultMoneyFormat</b> , que es una de las diversas máscaras que pueden utilizarse para crear la serie devuelta por la función <b>StrLib.formatNumber</b> .

Variable de sistema	Descripción
defaultNumericFormat	Especifica el valor de <b>defaultNumericFormat</b> , que es una de las diversas máscaras que pueden utilizarse para crear la serie devuelta por la función <b>StrLib.formatNumber</b> .
defaultTimeFormat	Especifica el valor de <b>defaultTimeFormat</b> , que es una de las diversas máscaras que pueden utilizarse para crear la serie devuelta por la función <b>StrLib.formatTime</b> .
defaultTimestampFormat	Especifica el valor de <b>defaultTimestampFormat</b> , que es una de las diversas máscaras que pueden utilizarse para crear la serie devuelta por la función <b>StrLib.formatTimestamp</b> .

La tabla siguiente muestra las constantes de sistema de la biblioteca **StrLib**. Todas son de tipo **STRING**.

Variable de sistema	Descripción
db2TimestampFormat	El patrón <i>aaaa-MM-dd-HH.mm.ss.ffffff</i> , que es el formato de indicación de la hora por omisión de IBM DB2.
eurDateFormat	El patrón <i>dd.MM.aaaa</i> , que es el formato de fecha estándar europeo de IBM.
eurTimeFormat	El patrón <i>HH.mm.ss</i> , que es el formato de hora estándar europeo de IBM.
isoDateFormat	El patrón <i>aaaa-MM-dd</i> , que es el formato de fecha especificado por International Standards Organization (ISO).
isoTimeFormat	El patrón <i>HH.mm.ss</i> , que es el formato de hora especificado por International Standards Organization (ISO).
jisDateFormat	El patrón <i>aaaa-MM-dd</i> , que es el formato de fecha estándar industrial japonés.
jisTimeFormat	El patrón <i>HH:mm:ss</i> , que es el formato de hora estándar industrial japonés.
odbcTimestampFormat	El patrón <i>aaaa-MM-dd HH:mm:ss.ffffff</i> , que es el formato de indicación de la hora de ODBC.
usaDateFormat	El patrón <i>MM/dd/aaaa</i> , que es el formato de fecha estándar para EE.UU. de IBM.
usaTimeFormat	El patrón <i>hh:mm AM</i> , que es el formato de hora estándar de EE.UU. de IBM.

#### Consulta relacionada

"formatDate()" en la página 877

"formatNumber()" en la página 878

"formatTime()" en la página 879

"formatTimeStamp()" en la página 880

## characterAsInt()

La función de formato de serie **StrLib.characterAsInt** convierte una serie de caracteres en una serie de enteros correspondiente al primer carácter de la expresión de caracteres.

```
StrLib.characterAsInt(texto STRING in)
returns (resultado INT)
```

*resultado*

Una variable de tipo INT.

*texto*

Un literal, una variable o una expresión que devuelve una serie de caracteres de tipo CHAR.

Para convertir una serie de enteros en una serie de caracteres, utilice la función de formato de serie **StrLib.integerAsChar**.

### Consulta relacionada

“Biblioteca StrLib de EGL” en la página 867

“integerAsChar()” en la página 883

## clip()

La función de formato de serie **StrLib.clip** suprime espacios en blanco finales y nulos del final de las series de caracteres devueltas.

```
StrLib.clip(texto STRING in)
returns (resultado STRING)
```

*resultado*

Una serie de caracteres.

*texto*

Un literal, una variable o una expresión que devuelve una serie de caracteres de tipo CHAR.

### Consulta relacionada

“Biblioteca StrLib de EGL” en la página 867

## compareStr()

La función de sistema **StrLib.compareStr** compara dos subseries de acuerdo con su orden ASCII o EBCDIC durante la ejecución.

```
StrLib.compareStr(
 destino VagText in,
 indiceSubserieDestino INT in,
 longitudSubserieDestino INT in,
 origen VagText in,
 indiceSubserieOrigen INT in,
 longitudSubserieOrigen INT in)
returns (resultado INT)
```

*resultado*

Elemento numérico que recibe uno de los siguientes valores (definido como de tipo INT o el equivalente: tipo BIN con longitud 9 y sin posiciones decimales) devueltos por la función:

- 1 La subserie basada en *destino* es menor que la subserie basada en *origen*
- 0 La subserie basada en *destino* es igual que la subserie basada en *origen*
- 1 La subserie basada en *destino* es mayor que la subserie basada en *origen*

*destino*

Serie de la que se deriva una subserie destino. Puede ser un elemento o un literal.

*índiceSubserieDestino*

Identifica el byte inicial de la subserie de *destino*, suponiendo que el primer byte de *destino* tenga el valor de índice 1. Este índice puede ser un literal entero. Como alternativa, este índice puede ser un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

*longitudSubserieDestino*

Identifica el número de bytes de la subserie que se deriva de *destino*. La longitud puede ser un literal entero. Como alternativa, este índice puede ser un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

*origen*

Serie de la que se deriva una subserie origen. Puede ser un elemento o un literal.

*índiceSubserieOrigen*

Identifica el byte inicial de la subserie de *origen*, suponiendo que el primer byte de *origen* tenga el valor de índice 1. Este índice puede ser un literal entero. Como alternativa, este índice puede ser un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

*longitudSubserieOrigen*

Identifica el número de bytes de la subserie que se deriva de *origen*. La longitud puede ser un literal entero. Como alternativa, este índice puede ser un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

Se realiza una comparación binaria de byte a byte de los valores de subserie. Si las subseries no tienen la misma longitud, la subserie más corta se rellena con espacios antes de la comparación.

**Consideraciones de definición:** En `sysVar.errorCode` se devuelven los siguientes valores:

- 8 El índice es menor que 1 o mayor que la longitud de la serie.
- 12 La longitud es menor que 1.
- 20 Índice de doble byte no válido. El índice de una serie DBCHAR o UNICODE señala a la mitad del carácter de doble byte
- 24 Longitud de doble byte no válida. La longitud en bytes de una serie DBCHAR o UNICODE es impar (las longitudes de doble byte deben ser siempre pares).

**Ejemplo:**

```
target = "123456";
source = "34";
result =
 StrLib.compareStr(target,3,2,source,1,2);
// resultado = 0
```

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca StrLib de EGL” en la página 867

### concatenate()

La función de sistema **StrLib.concatenate** concatena dos series.

```
StrLib.concatenate(
 destino VagText inOut,
 origen VagText in)
returns (resultado INT)
```

*resultado*

Elemento numérico que recibe uno de los siguientes valores (definido como de tipo INT o el equivalente: tipo BIN con longitud 9 y sin posiciones decimales) devueltos por la función:

- 1 La serie concatenada es demasiado larga para caber en el elemento destino y la serie se ha truncado, como se describe más adelante
- 0 La serie concatenada cabe en el elemento destino

*destino*

Elemento destino

*origen*

Literal o elemento origen

Cuando se concatenan dos series, se produce lo siguiente:

1. Los espacios finales o nulos se suprimen de la serie destino.
2. La serie origen se añade a la serie generada en el paso 1.
3. Si la serie generada en el paso 2 es más larga que el elemento de serie destino, la serie se trunca. Si es más corta que el elemento destino, se rellena con espacios en blanco.

### Ejemplo:

```
phrase = "and/ "; // CHAR(7)
or = "or";
result =
 StrLib.concatenate(phrase,or);
if (result == 0)
 print phrase; // phrase = "and/or "
end
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca StrLib de EGL” en la página 867

### concatenateWithSeparator()

La función de sistema **StrLib.concatenateWithSeparator** concatena dos series, insertando una serie separadora entre ellas. Si la longitud inicial de la serie destino es cero (sin contar los blancos finales y los nulos), el separador se omite y la serie origen se copia en la serie destino.

```
StrLib.concatenateWithSeparator(
 destino VagText inOut,
 origen VagText in,
 separador VagText in)
returns (resultado INT)
```

*resultado*

Elemento numérico que recibe uno de los siguientes valores (definido como de tipo INT o el equivalente: tipo BIN con longitud 9 y sin posiciones decimales) devueltos por la función

- 0 La serie concatenada cabe en el elemento destino.
- 1 La serie concatenada es demasiado larga para caber en el elemento destino y la serie se ha truncado, como se describe más adelante

*destino*

Elemento destino.

*origen*

Literal o elemento origen.

*separador*

Literal o elemento separador.

Los espacios y nulos finales se truncan en el *destino*; a continuación, las series *separador* y *origen* se añaden al final del valor truncado. Si la concatenación es más larga de lo que permite el destino, se produce un truncamiento. Si la concatenación es más corta de lo que permite el destino, el valor concatenado se rellena con espacios.

### Ejemplo:

```
phrase = "and"; // CHAR(7)
or = "or";
result =
 StrLib.concatenateWithSeparator(phrase,or,"/");
if (result == 0)
 print phrase; // phrase = "and/or "
end
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca StrLib de EGL” en la página 867

## copyStr()

La función de sistema **StrLib.copyStr** copia una subserie en otra.

```
StrLib.copyStr(
 destino VagText inOut,
 índiceSubserieDestino INT in,
 longitudSubserieDestino INT in,
 origen VagText in,
 índiceSubserieOrigen INT in,
 longitudSubserieOrigen INT in)
```

*destino*

Serie de la que se deriva una subserie destino. Puede ser un elemento o un literal.

*índiceSubserieDestino*

Identifica el byte inicial de *destino*, suponiendo que el primer byte de *destino* tiene el valor 1. Este índice puede ser un literal entero. Como alternativa, este índice puede ser un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

*longitudSubserieDestino*

Identifica el número de bytes de la subserie que se deriva de *destino*. La

longitud puede ser un literal entero. Como alternativa, la longitud puede ser un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

#### *origen*

Serie de la que se deriva una subserie origen. Puede ser un elemento o un literal.

#### *índiceSubserieOrigen*

Identifica el byte inicial de la subserie de *origen*, suponiendo que el primer byte de *origen* tiene el valor 1. Este índice puede ser un literal entero. Como alternativa, este índice puede ser un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

#### *longitudSubserieOrigen*

Identifica el número de bytes de la subserie que se deriva de *origen*. La longitud puede ser un literal entero. Como alternativa, la longitud puede ser un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

Si el origen es más largo que el destino, el origen se trunca. Si el origen es más corto que el destino, el valor origen se rellena a la derecha con espacios.

**Consideraciones de definición:** En `sysVar.errorCode` se devuelven los siguientes valores:

- 8 El índice es menor que 1 o mayor que la longitud de la serie.
- 12 La longitud es menor que 1.
- 20 Índice de doble byte no válido. El índice de una serie DBCHAR o UNICODE señala a la mitad del carácter de doble byte.
- 24 Longitud de doble byte no válida. La longitud en bytes de una serie DBCS o UNICODE es impar (las longitudes de doble byte deben ser siempre pares).

#### **Ejemplo:**

```
target = "120056";
source = "34";
StrLib.copyStr(target,3,2,source,1,2);
// destino = "123456"
```

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Biblioteca StrLib de EGL” en la página 867

## **defaultDateFormat**

La variable de sistema `StrLib.defaultDateFormat` especifica el valor de `defaultDateFormat`, que es una de las diversas máscaras que pueden utilizarse para crear la serie devuelta por la función `StrLib.formatDate`.

El valor inicial de `StrLib.defaultDateFormat` es el valor de la propiedad de entorno de ejecución Java `vgj.default.dateFormat`. Si no se establece dicha propiedad, el valor inicial de `StrLib.defaultDateFormat` es `MM/dd/aaaa`.

Para obtener detalles acerca de las características de una máscara horaria, consulte el apartado *Especificadores de fecha, hora e indicación de la hora*.

Tipo: STRING

#### **Consulta relacionada**

“Especificadores de fecha, hora e indicación de la hora” en la página 45

“Biblioteca StrLib de EGL” en la página 867

“formatDate()” en la página 877

“Propiedades de ejecución de Java (detalles)” en la página 540

### **defaultMoneyFormat**

La variable de sistema **StrLib.defaultMoneyFormat** especifica el valor de **defaultMoneyFormat**, que es una de las diversas máscaras que pueden utilizarse para crear la serie devuelta por la función **StrLib.formatNumber**.

El valor inicial de **StrLib.defaultMoneyFormat** es el valor de la propiedad de entorno de ejecución Java **vgj.default.moneyFormat**. Si no se establece dicha propiedad, el valor inicial de **StrLib.defaultMoneyFormat** es una serie vacía.

Para obtener detalles acerca de las características de una máscara numérica, consulte *formatNumber()*.

Tipo: STRING

#### **Consulta relacionada**

“Biblioteca StrLib de EGL” en la página 867

“formatNumber()” en la página 878

“Propiedades de ejecución de Java (detalles)” en la página 540

### **defaultNumericFormat**

La variable de sistema **StrLib.defaultNumericFormat** especifica el valor de **defaultNumericFormat**, que es una de las diversas máscaras que pueden utilizarse para crear la serie devuelta por la función **StrLib.formatNumber**.

El valor inicial de **StrLib.defaultNumericFormat** es el valor de la propiedad de entorno de ejecución Java **vgj.default.numericFormat**. Si no se establece dicha propiedad, el valor inicial de **StrLib.defaultNumericFormat** es una serie vacía.

Para obtener detalles acerca de las características de una máscara numérica, consulte *formatNumber()*.

Tipo: STRING

#### **Consulta relacionada**

“Biblioteca StrLib de EGL” en la página 867

“formatNumber()” en la página 878

“Propiedades de ejecución de Java (detalles)” en la página 540

### **defaultTimeFormat**

La variable de sistema **StrLib.defaultTimeFormat** especifica el valor de **defaultTimeFormat**, que es una de las diversas máscaras que pueden utilizarse para crear la serie devuelta por la función **StrLib.formatTime**. La variable no se utiliza en ningún otro contexto.

El valor inicial de **StrLib.defaultTimeFormat** es el valor de la propiedad de entorno de ejecución Java **vgj.default.timeFormat**. Si no se establece dicha propiedad, el valor inicial de **StrLib.defaultTimeFormat** es *HH:mm:ss*.

Para obtener detalles acerca de las características de una máscara horaria, consulte el apartado *Especificadores de fecha, hora e indicación de la hora*.

Tipo: STRING

#### Consulta relacionada

“Especificadores de fecha, hora e indicación de la hora” en la página 45

“Biblioteca StrLib de EGL” en la página 867

“formatTime()” en la página 879

“Propiedades de ejecución de Java (detalles)” en la página 540

### defaultTimestampFormat

La variable de sistema **StrLib.defaultTimestampFormat** especifica el valor de **defaultTimestampFormat**, que es una de las diversas máscaras que pueden utilizarse para crear la serie devuelta por la función **StrLib.formatTimestamp**.

El valor inicial de **StrLib.defaultTimestampFormat** es el valor de la propiedad de entorno de ejecución Java **vgj.default.timestampFormat**. Si no se establece dicha propiedad, el valor inicial de **StrLib.defaultTimestampFormat** es una serie vacía.

Para obtener detalles acerca de las características de una máscara de indicación de la hora, consulte el apartado *Especificadores de fecha, hora e indicación de la hora*.

Tipo: STRING

#### Consulta relacionada

“Especificadores de fecha, hora e indicación de la hora” en la página 45

“Biblioteca StrLib de EGL” en la página 867

“formatTimeStamp()” en la página 880

“Propiedades de ejecución de Java (detalles)” en la página 540

### findStr()

La función de sistema **StrLib.findStr** busca la primera aparición de una subserie dentro de una serie.

```
StrLib.findStr(
 origen VagText in,
 indiceSubserieOrigen INT inOut,
 longitudSubserieOrigen INT in,
 serieBúsqueda VagText in)
returns (resultado INT)
```

*resultado*

Elemento numérico que recibe uno de los siguientes valores (definido como de tipo INT o el equivalente: tipo BIN con longitud 9 y sin posiciones decimales) devueltos por la función:

-1 No se ha encontrado la serie de búsqueda

0 Se ha encontrado la serie de búsqueda

*origen*

Serie de la que se deriva una subserie origen. Puede ser un elemento o un literal.

### *índiceSubserieOrigen*

Identifica el byte inicial de la subserie de *origen*, suponiendo que el primer byte de *origen* tenga el valor de índice 1. Este índice puede ser un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

### *longitudSubserieOrigen*

Identifica el número de bytes de la subserie que se deriva de *origen*. Este índice puede ser un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

### *serieBúsqueda*

Literal o elemento de serie que debe buscarse en la subserie origen. Los blancos o nulos finales se truncan en la serie de búsqueda antes de que ésta se inicie.

Si *serieBúsqueda* se encuentra en la subserie origen, *índiceSubserieOrigen* se establece para indicar su ubicación (el byte de la subserie origen donde empieza la subserie coincidente). En caso contrario, *índiceSubserieOrigen* no cambia.

**Consideraciones de definición:** Se devuelven los siguientes valores en `sysVar.errorCode`:

- 8 El índice es menor que 1 o mayor que la longitud de la serie.
- 12 La longitud es menor que 1.
- 20 Índice de doble byte no válido. El índice de una serie DBCHAR o UNICODE señala a la mitad del carácter de doble byte.
- 24 Longitud de doble byte no válida. La longitud en bytes de una serie DBCHAR o UNICODE es impar (las longitudes de doble byte deben ser siempre pares).

### **Ejemplo:**

```
source = "123456";
sourceIndex = 1
sourceLength = 6
search = "34";
result =
 StrLib.findStr(source,sourceIndex,sourceLength,"34");
// resultado = 0, sourceIndex = 3
```

### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

### **Consulta relacionada**

“Biblioteca StrLib de EGL” en la página 867

### **formatDate()**

La función de sistema **StrLib.formatDate** da formato a un valor de fecha y devuelve un valor de tipo STRING.

```
StrLib.formatDate(
 valorFecha DATE in
 [, formatoFecha STRING in])
returns (resultado STRING)
```

### *resultado*

Una variable de tipo STRING.

*valorFecha*

El valor que debe formatearse. Puede ser cualquier expresión que dé como resultado un valor de fecha; por ejemplo, la variable de sistema **VGVar.currentGregorianCalendar**.

*formatoFecha*

Identifica el formato de fecha, como se describe en el apartado *Especificadores de fecha, hora e indicación de la hora*. Si no especifica ningún valor para *formatoFecha*, el entorno de ejecución de EGL utiliza el formato de fecha del entorno local Java.

Puede utilizar una serie, la variable de sistema **StrLib.defaultDateFormat** (como se describe en *defaultDateFormat*) o cualquiera de estas constantes:

**StrLib.eurDateFormat**

El patrón *dd.MM.aaaa*, que es el formato de fecha estándar europeo de IBM.

**StrLib.isoDateFormat**

El patrón *aaaa-MM-dd*, que es el formato de fecha especificado por International Standards Organization (ISO).

**StrLib.jisDateFormat**

El patrón *aaaa-MM-dd*, que es el formato de fecha estándar industrial japonés.

**StrLib.usaDateFormat**

El patrón *MM/dd/aaaa*, que es el formato de fecha estándar para EE.UU. de IBM.

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“DATE” en la página 41

“Especificadores de fecha, hora e indicación de la hora” en la página 45

“defaultDateFormat” en la página 874

“Biblioteca StrLib de EGL” en la página 867

**formatNumber()**

La función de serie **StrLib.formatNumber** devuelve un número como serie con formato.

```
StrLib.formatNumber(
 expresiónNumérica anyNumericExpression in,
 formatoNumérico STRING in)
returns (resultado STRING)
```

*resultado*

Una variable de tipo STRING.

*expresiónNumérica*

El valor numérico que debe formatearse. Puede ser cualquier expresión que dé como resultado un número.

*formatoNumérico*

Una serie que define cómo debe formatearse el número. En la siguiente tabla encontrará los detalles. La serie es obligatoria, pero puede utilizar la variable de sistema **StrLib.defaultMoneyFormat** o **StrLib.defaultNumericFormat**. Para obtener detalles acerca de esas variables, consulte *defaultMoneyFormat* y *defaultNumericFormat*.

Los caracteres válidos son los siguientes:

- # Un espacio reservado para un dígito.
- \* Utilice un asterisco (\*) como carácter de relleno para un cero inicial.
- & Utilice un cero como carácter de relleno para un cero inicial.
- # Utilice un espacio como carácter de relleno para un cero inicial.
- < Justifique a la izquierda el número.
- , Utilice un separador numérico dependiente del entorno local a menos que la posición contenga un cero inicial.
- . Utilice una coma decimal dependiente del entorno local.
- Utilice un signo menos (-) para los valores menores que 0; utilice un espacio para los valores mayores o iguales que 0.
- + Utilice un signo menos para los valores menores que 0; utilice un signo más (+) para los valores mayores o iguales que 0.
- ( Preceda los valores negativos con un paréntesis izquierdo , según sea apropiado en la contabilidad.
- ) Coloque un paréntesis derecho después de un valor negativo, según sea apropiado en la contabilidad.
- \$ Preceda el valor con un símbolo de moneda dependiente del entorno local.
- @ Coloque el símbolo de moneda dependiente del entorno local después del valor.

#### Consulta relacionada

“defaultMoneyFormat” en la página 875

“defaultNumericFormat” en la página 875

“Biblioteca StrLib de EGL” en la página 867

### formatTime()

La función de fecha y hora **StrLib.formatTime** da formato a un valor de hora y devuelve un valor de tipo STRING.

```
StrLib.formatTime(
 aTime Time in
 [, formatoHora STRING in
]
)
returns (resultado STRING)
```

*resultado*

Una variable de tipo STRING.

*aTime*

El valor que debe formatearse. Puede ser cualquier expresión que dé como resultado un valor de hora; por ejemplo, la variable de sistema

**DateTimeLib.currentTime**.

*formatoHora*

Identifica el formato de hora, como se describe en el apartado *Especificadores de fecha, hora e indicación de la hora*. Si no especifica ningún valor para *formatoHora*, el entorno de ejecución de EGL utiliza el formato de hora del entorno local Java.

Puede utilizar una serie, la variable de sistema **StrLib.defaultTimeFormat** (como se describe en *defaultTimeFormat*) o cualquiera de estas constantes:

**eurTimeFormat**

El patrón *HH.mm.ss*, que es el formato de hora estándar europeo de IBM.

**isoTimeFormat**

El patrón *HH.mm.ss*, que es el formato de hora especificado por International Standards Organization (ISO).

**jisTimeFormat**

El patrón *HH:mm:ss*, que es el formato de hora estándar industrial japonés.

**usaTimeFormat**

El patrón *hh:mm AM*, que es el formato de hora estándar de EE.UU. de IBM.

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Especificadores de fecha, hora e indicación de la hora” en la página 45

“defaultTimeFormat” en la página 875

“Biblioteca StrLib de EGL” en la página 867

“TIME” en la página 43

**formatTimeStamp()**

La función de formato de fecha y hora **StrLib.formatTimeStamp** da a un parámetro el formato de un valor de fecha y hora y devuelve un valor de tipo STRING.

```
StrLib.formatTimeStamp(
 aTimeStamp TimeStamp in
 [, formatoIndicaciónHora STRING in
]
)
returns (resultado STRING)
```

*resultado*

Una variable de tipo STRING.

*aTimeStamp*

El valor TIMESTAMP que debe formatearse. Puede ser cualquier expresión que dé como resultado un valor TIMESTAMP; por ejemplo, la variable de sistema **DateTimeLib.currentTimeStamp**.

*formatoIndicaciónHora*

Identifica el formato de fecha, como se describe en el apartado *Especificadores de fecha, hora e indicación de la hora*.

Puede utilizar una serie, la variable de sistema

**StrLib.defaultTimestampFormat** (como se describe en *defaultTimestampFormat*) o cualquiera de estas constantes:

**db2TimeStampFormat**

El patrón *aaaa-MM-dd-HH.mm.ss.ffffff*, que es el formato de indicación de la hora por omisión de IBM DB2.

**odbcTimeStampFormat**

El patrón *aaaa-MM-dd HH:mm:ss.ffffff*, que es el formato de indicación de la hora de ODBC.

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

"currentTimeStamp()" en la página 794

"Especificadores de fecha, hora e indicación de la hora" en la página 45

"defaultTimestampFormat" en la página 876

"Biblioteca StrLib de EGL" en la página 867

"TIMESTAMP" en la página 44

### getNextToken()

La función de sistema **StrLib.getNextToken** busca un símbolo en una subserie y copia dicho símbolo en un elemento destino.

Los símbolos son series separadas por caracteres delimitadores. Por ejemplo, si el espacio entre caracteres (" ") y la coma (",") se definen como delimitadores, la serie "CALL PROGRAM ARG1,ARG2,ARG3" puede descomponerse en cinco símbolos "CALL", "PROGRAM", "ARG1", "ARG2" y "ARG3".

```
StrLib.getNextToken(
 destino VagText inOut,
 origen VagText in,
 índiceSubserieOrigen INT inOut,
 longitudSubserieOrigen INT inOut,
 delimitadorCarácter VagText in)
returns (resultado INT)
```

#### resultado

Un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales. El valor es uno de los siguientes:

- +n Número de caracteres del símbolo. El símbolo se copia de la subserie bajo revisión al elemento destino.
- 0 No había ningún símbolo en la subserie bajo revisión.
- 1 El símbolo se ha truncado cuando se ha copiado en el elemento destino.

#### destino

Elemento destino de tipo CHAR, DBCHAR, HEX, MBCHAR o UNICODE.

#### origen

Elemento origen de tipo CHAR, DBCHAR, HEX, MBCHAR o UNICODE. Puede ser un literal de cualquiera de esos tipos que no sea UNICODE.

#### índiceSubserieOrigen

Identifica el byte inicial en el que se debe empezar a buscar un delimitador, suponiendo que el primer byte de *origen* tiene el valor 1. *índiceSubserieOrigen* puede ser un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales. Si se encuentra un símbolo, el valor de *índiceSubserieOrigen* se cambia por el índice del primer carácter que sigue al símbolo.

#### longitudSubserieOrigen

Indica el número de bytes de la subserie que se está revisando.

*longitudSubserieOrigen* puede ser un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales. Si se encuentra un símbolo, el valor de *longitudSubserieOrigen* se cambia por el número de bytes de la subserie que empieza después del símbolo devuelto.

#### delimitadorCarácter

Uno o más caracteres delimitadores, sin caracteres que los separen entre sí.

Puede ser un elemento de tipo CHAR, DBCHAR, HEX, MBCHAR o UNICODE. Puede ser un literal de cualquiera de esos tipos que no sea UNICODE.

Puede invocar una secuencia de llamadas para recuperar cada símbolo de una subserie sin restablecer los valores para *índiceSubserieOrigen* y *longitudSubserieOrigen*, como se muestra más adelante en un ejemplo.

**Condiciones de error:** Se devuelven los siguientes valores en **SysVar.errorCode**:

- 8 *índiceSubserieOrigen* es menor que 1 o es mayor que el número de bytes de la subserie que se está revisando.
- 12 *longitudSubserieOrigen* es menor que 1.
- 20 El valor de *índiceSubserieOrigen* de una serie DBCHAR o UNICODE hace referencia al medio de un carácter de doble byte.
- 24 El valor de *longitudSubserieOrigen* de una serie DBCHAR o UNICODE es impar (las longitudes de doble byte deben ser siempre pares).

### Ejemplo:

```
Function myFunction()
 myVar myStructurePart;
 myRecord myRecordPart;

 i = 1;
 myVar.mySourceSubstringIndex = 1;
 myVar.mySourceSubstringLength = 29;

 while (myVar.mySourceSubstringLength > 0)
 myVar.myResult = StrLib.getNextToken(myVar.myTarget[i],
 "CALL PROGRAM arg1, arg2, arg3",
 myVar.mySourceSubstringIndex,
 myVar.mySourceSubstringLength, " ,");

 if (myVar.myResult > 0)
 myRecord.outToken = myVar.myTarget[i];
 add myRecord;
 set myRecord empty;
 i = i + 1;
 end
 end
end

Record myStructurePart
 01 myTarget CHAR(80)[5];
 01 mySource CHAR(80);
 01 myResult myBinPart;
 01 mySourceSubstringIndex INT;
 01 mySourceSubstringLength BIN(9,0);
 01 i myBinPart;
end

Record myRecordPart
 serialRecord:
 fileName="Output"
 end
 01 outToken CHAR(80);
end
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca StrLib de EGL” en la página 867

## integerAsChar()

La función de formato de serie **StrLib.integerAsChar** convierte una serie de enteros en una serie de caracteres.

```
StrLib.integerAsChar(entero INT in)
returns (resultado STRING)
```

*resultado*

Una variable de tipo STRING.

*entero*

Un literal, una variable o una expresión que devuelve un entero de tipo BIGINT, INT o SMALLINT.

Para convertir una serie de caracteres en una serie de enteros, utilice la función de formato de serie **StrLib.characterAsInt**.

### Consulta relacionada

“Biblioteca StrLib de EGL” en la página 867

“characterAsInt()” en la página 870

## lowerCase()

La función de formato de serie **StrLib.lowerCase** convierte todos los valores en mayúsculas de una serie de caracteres en valores en minúsculas. Los valores numéricos y en minúsculas existentes no se ven afectados.

```
StrLib.lowerCase(texto STRING in)
returns (resultado STRING)
```

*resultado*

Una variable de tipo STRING.

*texto*

Un literal, una variable o una expresión que devuelve una serie de caracteres de tipo CHAR.

La función **StrLib.lowerCase** no tiene efecto sobre los caracteres de doble byte en elementos de tipo DBCHAR o MBCHAR.

Para convertir valores en minúsculas en valores en mayúsculas, utilice la función de formato de serie **StrLib.upperCase**.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca StrLib de EGL” en la página 867

“upperCase()” en la página 886

## setBlankTerminator()

La función de sistema **StrLib.setBlankTerminator** cambia un terminador nulo y los caracteres subsiguientes por espacios. **StrLib.setBlankTerminator** cambia un valor de serie devuelto desde un programa C o C++ por un valor de carácter que puede operar correctamente en un programa EGL.

```
StrLib.setBlankTerminator(destino VagText inOut)
```

*destino*

El elemento de serie destino. Si no se encuentra ningún nulo en *serieDestino*, la función no tiene ningún efecto.

**Ejemplo:**

```
StrLib.setBlankTerminator(target);
```

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca StrLib de EGL” en la página 867

**setNullTerminator()**

La función de sistema **StrLib.setNullTerminator** cambia todos los espacios finales de una serie por nulos. Puede utilizar **StrLib.setNullTerminator** para convertir un elemento antes de pasarlo a un programa C o C++ que espera una serie terminada en nulo como argumento.

```
StrLib.setNullTerminator(destino VagText inOut)
```

*destino*

Serie que debe convertirse

Se buscan los espacios finales y nulos en la serie destino. Los espacios encontrados se cambian por nulos.

**Consideraciones de definición:** Puede devolverse el siguiente valor en **sysVar.errorCode**:

16 El último byte de la serie no es un espacio ni un nulo

**Ejemplo:**

```
StrLib.setNullTerminator(myItem01);
```

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca StrLib de EGL” en la página 867

**setSubStr()**

La función de sistema **StrLib.setSubStr** sustituye cada carácter de una subserie por un carácter especificado.

```
StrLib.setSubStr(
 destino VagText inOut,
 índiceSubserieDestino INT in,
 longitudSubserieDestino INT in,
 origen)
```

*destino*

Elemento que se cambia.

*índiceSubserieDestino*

Identifica el byte inicial de la subserie de *destino*, suponiendo que el primer byte de *destino* tenga el valor de índice 1. Este índice puede ser un literal entero. Como alternativa, este índice puede ser un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

### *longitudSubserieDestino*

Identifica el número de bytes de la subserie que se deriva de *destino*. La longitud puede ser un literal entero. Como alternativa, la longitud puede ser un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

### *origen*

Si el elemento destino es CHAR, MBCHAR o HEX, el elemento origen debe ser un elemento CHAR, MBCHAR o HEX de un byte o un literal CHAR. Si el destino es un elemento DBCHAR o UNICODE, el origen debe ser un elemento DBCHAR o UNICODE de un solo carácter.

**Consideraciones de definición:** Se devuelven los siguientes valores en SysVar.errorCode:

- 8 El índice es menor que 1 o mayor que la longitud de la serie
- 12 La longitud es menor que 1
- 20 Índice de doble byte no válido. El índice de una serie DBCHAR o UNICODE señala a la mitad del carácter de doble byte
- 24 Longitud de doble byte no válida. La longitud en bytes de una serie DBCHAR o UNICODE es impar (las longitudes de doble byte deben ser siempre pares)

### **Ejemplo:**

```
StrLib.setSubStr(target,12,5," ");
```

### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

### **Consulta relacionada**

“Biblioteca StrLib de EGL” en la página 867

## **spaces()**

La función de sistema **StrLib.spaces** devuelve una serie compuesta de un número especificado de espacios.

```
StrLib.spaces(cuentaCaracteres INT in)
returns (resultado STRING)
```

### *resultado*

Una variable de tipo STRING.

### *cuentaCaracteres*

La longitud de la serie de espacios a devolver.

### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

### **Consulta relacionada**

“Biblioteca StrLib de EGL” en la página 867

## **strLen()**

La función de sistema **StrLib.strLen** devuelve el número de bytes de un elemento, excluyendo los espacios finales y los nulos.

```
StrLib.strLen(origen VagText in)
returns (resultado INT)
```

*resultado*

Un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

*origen*

Literal o elemento de serie que debe medirse.

**Ejemplo:**

```
length = StrLib.strLen(source);
```

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca StrLib de EGL” en la página 867

**textLen()**

La función de sistema **StrLib.textLen** devuelve el número de caracteres de una expresión de texto, excluyendo los espacios finales y los nulos.

```
StrLib.textLen(origen STRING in)
returns (resultado INT)
```

*resultado*

Un elemento definido como de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

*origen*

La expresión de texto en cuestión.

**Ejemplo:**

```
length = StrLib.textLen(source);
```

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

“Biblioteca StrLib de EGL” en la página 867

“Expresiones de texto” en la página 505

**upperCase()**

La función de formato de serie **StrLib.upperCase** convierte todos los valores en minúsculas de una serie de caracteres en valores en mayúsculas. Los valores numéricos y en mayúsculas existentes no se ven afectados.

```
StrLib.upperCase(texto STRING in)
returns (resultado STRING)
```

*resultado*

Una variable de tipo STRING.

*texto*

Un literal, una variable o una expresión que devuelve una serie de caracteres de tipo CHAR.

La función **StrLib.upperCase** no tiene efecto sobre los caracteres de doble byte en elementos de tipo DBCHAR o MBCHAR.

Para convertir una serie de caracteres a minúsculas, utilice la función de formato de serie **StrLib.lowerCase**.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca StrLib de EGL” en la página 867

“lowerCase()” en la página 883

## Biblioteca SysLib de EGL

Función	Descripción
<code>beginDatabaseTransaction([baseDatos])</code>	Empieza con una transacción de base de datos relacional, pero sólo cuando el tiempo de ejecución de EGL no esté comprometiendo los cambios automáticamente.
<code>result = bytes(field)</code>	Devuelve el número de bytes en un área de memoria con nombre.
<code>calculateChkDigitMod10 (texto, longitudComprobación, resultado)</code>	Coloca un dígito de comprobación Modulus-10 en un elemento de caracteres que empieza por una serie de enteros.
<code>calculateChkDigitMod11 (texto, longitudComprobación, resultado)</code>	Coloca un dígito de comprobación Modulus-11 en un elemento de caracteres que empieza por una serie de enteros.
<code>callCmd (serieMandato[, serieModalidad])</code>	Ejecuta un mandato del sistema y espera a que éste finalice.
<code>commit()</code>	Guarda actualizaciones realizadas en bases de datos, colas de mensajes de MQSeries y archivos recuperables de CICS desde el último compromiso. Un programa o envoltura Java generado también guarda las actualizaciones efectuadas por un programa remoto COBOL basado en CICS (incluidas las actualizaciones realizadas en los archivos recuperables CICS), pero sólo cuando la llamada al programa remoto COBOL implica una unidad de trabajo controlada por el cliente, como se describe en el apartado <i>luwControl del elemento callLink</i> .
<code>result = conditionAsInt (booleanExpression)</code>	Acepta una expresión lógica (como por ejemplo <code>myVar == 6</code> ), devolviendo un 1 si la expresión es true y un 0 si la expresión es false.
<code>connect (baseDatos, idUsuario, contraseña[, ámbitoCompromiso[, opciónDesconexión[, nivelAislamiento[, controlCompromiso]]]])</code>	Cierra todos los cursores, libera bloqueos, finaliza cualquier conexión existente y se conecta a la base de datos.
<code>convert (destino, sentido, tablaConversión)</code>	Convierte datos entre los formatos EBCDIC (sistema principal) y ASCII (estación de trabajo) o bien realiza la conversión de página de códigos dentro de un único formato.
<code>defineDatabaseAlias (alias, baseDatos)</code>	Crea un alias que puede utilizarse para establecer una conexión nueva con una base de datos a la que el código ya está conectado.

Función	Descripción
<code>disconnect ([baseDatos])</code>	Desconecta de la base de datos especificada o (si no se ha especificado ninguna base de datos) de la base de datos actual.
<code>disconnectAll ()</code>	Desconecta de todas las bases de datos conectadas actualmente.
<code>errorLog ()</code>	Copia texto en las anotaciones de error que ha iniciado la función de sistema <b>SysLib.startLog</b> .
<code>result = getCmdLineArg (index)</code>	Devuelve el argumento especificado de la lista de argumentos con los que se invocó el programa EGL. El argumento especificado se devuelve como un valor de serie.
<code>result = getCmdLineArgCount ()</code>	Devuelve el número de argumentos que se han utilizado para iniciar el programa EGL principal.
<code>result = getMessage (key [, insertArray])</code>	Devuelve un mensaje del archivo al que se hace referencia en la propiedad Java de tiempo de ejecución <code>vgj.message.file</code> .
<code>result = getProperty(propertyName)</code>	Recupera el valor de una propiedad Java de tiempo de ejecución. Si no se encuentra la propiedad especificada, la función devuelve una serie nula ( <code>""</code> ).
<code>loadTable (nombreArchivo, insertarEnCláusula[, delimitador])</code>	Carga información de un archivo en una base de datos relacional.
<code>result = maximumSize (arrayName)</code>	Devuelve el número máximo de filas que pueden estar en una matriz dinámica de elementos de datos o registros; específicamente, la función devuelve el valor de la propiedad de matriz <b>maxSize</b> .
<code>queryCurrentDatabase (producto, release)</code>	Devuelve el número de producto y release de la base de datos conectada actualmente.
<code>rollback ()</code>	Invierte las actualizaciones realizadas en bases de datos y colas de mensajes de MQSeries desde el último compromiso. Esta inversión se produce en cualquier aplicación generada por EGL.
<code>setCurrentDatabase (baseDatos)</code>	Hace que la base de datos especificada sea la base de datos actualmente activa.
<code>setError (elementoEnError, claveMsj[, elementoInserción])</code> <code>setError (this, claveMsj[, elementoInserción])</code> <code>setError (textoMsj)</code>	Asocia un mensaje a un elemento de un PageHandler o un registro de UI o al PageHandler o registro de UI como un todo. El mensaje se coloca en la ubicación de un mensaje o código de mensajes JSF del JSP y se visualiza cuando lo hace la página Web relacionada.
<code>setLocale (códigoIdioma, códigoPaís[, variante])</code>	Se utiliza en PageHandlers y en programas que se ejecutan en una aplicación Web.
<code>setRemoteUser (idUserario, contraseña)</code>	Establece el ID de usuario y la contraseña que se utilizan en las llamadas a programas remotos de programas Java.

Función	Descripción
<code>result = size (arrayName)</code>	Devuelve el número de filas de la tabla de datos especificada o el número de elementos de la matriz especificada. La matriz puede ser una matriz de elementos de estructura, una matriz estática de elementos de datos o registros o una matriz dinámica de elementos de datos o registros.
<code>startCmd (serieMandato[, serieModalidad])</code>	Ejecuta un mandato del sistema y no espera a que éste finalice.
<code>startLog (archivoAnotaciones)</code>	Abre un archivo de anotaciones de error. El texto se escribe en dichas anotaciones cada vez que el programa invoca <b>SysLib.errorLog</b> .
<code>startTransaction (idTerm[, idImpr[, idTerm]])</code>	Invoca un programa principal de forma asíncrona, asocia dicho programa a una impresora o dispositivo de terminal y pasa un registro. Si EGL genera el programa receptor, el registro se utiliza para inicializar el registro de entrada; si VisualAge Generator produce el receptor, el registro se utiliza para inicializar el almacenamiento de trabajo.
<code>unloadTable (nombreArchivo, sentenciaSelect[, delimitador])</code>	Descarga información de una tabla de base de datos relacional en un archivo.
<code>verifyChkDigitMod10 (entrada, longitudComprobación, resultado)</code>	Verifica un dígito de comprobación Modulus-10 en un elemento de caracteres que empieza con una serie de enteros.
<code>verifyChkDigitMod11 (entrada, longitudComprobación, resultado)</code>	Verifica un dígito de comprobación Modulus-11 en un elemento de caracteres que empieza con una serie de enteros.
<code>wait (tiempoEnSegundos0)</code>	Suspende la ejecución durante el número especificado de segundos.

### **beginDatabaseTransaction()**

La función del sistema **SysLib.beginDatabaseTransaction** empieza con una transacción de base de datos relacional, pero solo cuando el tiempo de ejecución de EGL no esté comprometiendo los cambios automáticamente. Si los cambios se comprometen automáticamente, la función no surte efecto.

```
SysLib.beginDatabaseTransaction(
 [baseDatos STRING in])
```

*baseDatos*

Un nombre de base de datos especificado en `SysLib.connect` o `VGLib.connectionService`. Utilice un literal o variable de un tipo de carácter.

Si no especifica una conexión, la función afecta a la conexión actual.

Cuando invoca **SysLib.beginDatabaseTransaction**, una transacción empieza en la operación de E/S siguiente que utiliza la conexión especificada y la transacción finaliza cuando tiene lugar un compromiso o una retroacción, tal como se describe en la *Unidad de trabajo lógica*. Después del compromiso o la retroacción, el tiempo de ejecución EGL reanuda el compromiso automático de los cambios.

Para obtener detalles sobre compromisos automáticos, consulte *SysLib.connect* y *sqlCommitControl*.

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“Unidad lógica de trabajo” en la página 307

“Soporte de SQL” en la página 229

#### Consulta relacionada

“sqlCommitControl” en la página 400

“connect()” en la página 895

“connectionService()” en la página 916

### bytes()

La función de sistema **SysLib.bytes** devuelve el número de bytes de un área de memoria con nombre.

```
SysLib.bytes(campo fixedFieldOrArray in)
returns (resultado INT)
```

*resultado*

Un elemento numérico que recibe el número de bytes de *campo*. Hay dos casos a destacar:

- Si *campo* es una matriz, la función devuelve el número de bytes de un elemento
- Si *campo* es un registro SQL, la función devuelve el número de bytes del registro, incluidos los bytes sobrantes; para obtener información detallada, consulte la sección *Componentes internos de un registro SQL*

*campo*

Una matriz, elemento o registro

#### Example():

```
result = SysLib.bytes(myItem);
```

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

#### Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

“Tipos primitivos” en la página 34

“Diseño interno de los registros SQL” en la página 747

### calculateChkDigitMod10()

La función de sistema **SysLib.calculateChkDigitMod10** coloca un dígito de comprobación Modulus-10 en un elemento de carácter que empieza por una serie de enteros.

```
SysLib.calculateChkDigitMod10(
 texto anyChar inOut,
 longitudComprobación SMALLINT in,
 resultado SMALLINT inOut)
```

*texto*

Un elemento de carácter que empieza por una serie de enteros. El elemento debe incluir una posición adicional para el dígito de comprobación, situado inmediatamente a la derecha de los demás enteros.

*longitudComprobación*

Un elemento que contiene el número de caracteres que desea utilizar del

elemento *texto*, incluida la posición utilizada para el dígito de comprobación. Este elemento tiene 4 dígitos y es de tipo SMALLINT o BIN, sin posiciones decimales.

#### *resultado*

Un elemento que recibe un de dos valores:

- 0, si se ha creado el dígito de comprobación
- 1, si no se ha creado el dígito de comprobación

Este elemento tiene 4 dígitos y es de tipo SMALLINT o BIN, sin posiciones decimales.

Puede utilizar **SysLib.calculateChkDigitMod10** en una sentencia de invocación de función.

**Ejemplo:** En el ejemplo siguiente, myInput es un elemento de tipo CHAR y contiene el valor 1734289; myLength es un elemento de tipo SMALLINT y contiene el valor 7; y myResult es un elemento de tipo SMALLINT:

```
SysLib.verifyChkDigitMod10 (myInput, myLength, myResult);
```

Se utiliza un algoritmo para derivar el dígito de comprobación Modulus-10 y en ningún caso se tiene en cuenta el número de la posición del dígito de comprobación. El algoritmo se describe en relación a los valores de ejemplo:

1. Multiplique la posición de unidades del número de entrada por 2 y multiplique todas las posiciones alternativas, de derecha a izquierda, por 2:

$$8 \times 2 = 16$$

$$4 \times 2 = 8$$

$$7 \times 2 = 14$$

2. Añada los dígitos de los productos (16814) a los dígitos del número de entrada (132) que no se hayan multiplicado por 2:

$$1 + 6 + 8 + 1 + 4 + 1 + 3 + 2 = 26$$

3. Para obtener el dígito de comprobación, sustraiga la suma del siguiente número más alto que termine en 0:

$$30 - 26 = 4$$

Si la sustracción produce 10, el dígito de comprobación será 0.

En este ejemplo, los caracteres originales de myInput pasan a ser los siguientes:

1734284

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Biblioteca SysLib de EGL” en la página 887

### **calculateChkDigitMod11()**

La función de sistema **SysLib.calculateChkDigitMod11** coloca un dígito de comprobación Modulus-11 en un elemento de carácter que empieza por una serie de enteros.

```
SysLib.calculateChkDigitMod11(
 texto anyChar inOut,
 longitudComprobación SMALLINT in,
 resultado SMALLINT inOut)
```

#### *texto*

Un elemento de carácter que empieza por una serie de enteros. El elemento

debe incluir una posición adicional para el dígito de comprobación, situado inmediatamente a la derecha de los demás enteros.

#### *longitudComprobación*

Un elemento que contiene el número de caracteres que desea utilizar del elemento *texto*, incluida la posición utilizada para el dígito de comprobación. Este elemento tiene 4 dígitos y es de tipo SMALLINT o BIN, sin posiciones decimales.

#### *resultado*

Un elemento que recibe un de dos valores:

- 0, si se ha creado el dígito de comprobación
- 1, si no se ha creado el dígito de comprobación

Este elemento tiene 4 dígitos y es de tipo SMALLINT o BIN, sin posiciones decimales.

Puede utilizar **SysLib.calculateChkDigitMod11** en una sentencia de invocación de función.

**Ejemplo:** En el ejemplo siguiente, myInput es un elemento de tipo CHAR y contiene el valor 56621869; myLength es un elemento de tipo SMALLINT y contiene el valor 8; y myResult es un elemento de tipo SMALLINT:

```
SysLib.verifyChkDigitMod (myInput, myLength, myResult);
```

Se utiliza un algoritmo para derivar el dígito de comprobación Modulus-11 y en ningún caso se tiene en cuenta el número de la posición del dígito de comprobación. El algoritmo se describe en relación a los valores de ejemplo:

1. Multiplique el dígito de la posición de unidades del número de entrada por 2, el de la posición de decenas por 3, el de la posición de centenas por 4, y así sucesivamente, pero deje que myLength " 1 sea el número mayor utilizado como multiplicador; y, si el número de entrada contiene más dígitos, inicie de nuevo la secuencia utilizando 2 como multiplicador:

```
6 x 2 = 12
8 x 3 = 24
1 x 4 = 4
2 x 5 = 10
6 x 6 = 36
6 x 7 = 42
5 x 2 = 10
```

2. Suma los productos del primer paso y divide la suma por 11:

```
(12 + 24 + 4 + 10 + 36 + 42 + 10) / 11
= 138 / 11
= 12 resto 6
```

3. Para obtener el dígito de comprobación, sustraiga el resto de 11 para obtener el dígito de autocombprobación:

```
11 - 6 = 5
```

Si el resto es 0 o 1, el dígito de comprobación será 0.

En este ejemplo, los caracteres originales de myInput pasan a ser los siguientes:

```
56621865
```

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Biblioteca SysLib de EGL” en la página 887

## callCmd()

La función del sistema **SysLib.callCmd** ejecuta un mandato del sistema y espera hasta que termina el mandato.

```
SysLib.callCmd(
 serieMandato STRING in
 [, serieModalidad STRING in
]
)
```

*serieMandato*

Identifica el mandato del sistema operativo que se desea invocar.

*serieModalidad*

La *serieModalidad* puede ser cualquier carácter o elemento de serie. El elemento puede estar en cualquiera de las dos modalidades:

- *formulario*: en la que cualquiera carácter de entrada queda a disposición del programa una vez teclado, es decir, cada pulsación de tecla se pasa directamente al mandato especificado.
- *línea*: en la que la entrada no está disponible hasta que se utiliza la tecla de carácter de línea nueva, es decir, no se envía ninguna información al mandato especificado hasta que se pulsa la tecla INTRO y entonces se envía toda la línea al mandato.

El mandato del sistema que se está ejecutando debe ser visible para el programa en ejecución. Por ejemplo, si ejecuta **callCmd("mySpecialProgram.exe")**, el programa "mySpecialProgram.exe" debe estar en un directorio incluido en la variable de entorno PATH. También puede especificar la ubicación completa del directorio, por ejemplo **callCmd("program files/myWork/mySpecialProgram.exe")**.

La función **SysLib.callCmd** sólo está soportada en entornos Java.

Utilice la función **SysLib.startCmd** para ejecutar un mandato del sistema que no espera hasta que finaliza el mandato.

### Conceptos relacionados

"Diagrama de sintaxis para funciones EGL" en la página 754

### Consulta relacionada

"Biblioteca SysLib de EGL" en la página 887

"startCmd()" en la página 910

## commit()

La función de sistema **SysLib.commit** guarda las actualizaciones efectuadas en bases de datos y colas de mensajes MQSeries desde la última operación de compromiso. Un programa o envoltura Java generado también guarda las actualizaciones efectuadas por un programa remoto COBOL basado en CICS (incluidas las actualizaciones realizadas en los archivos recuperables CICS), pero sólo cuando la llamada al programa remoto COBOL implica una unidad de trabajo controlada por el cliente, como se describe en el apartado *luwControl del elemento callLink*.

```
SysLib.commit()
```

En la mayoría de casos, EGL realiza un compromiso de una fase que afecta por turnos a cada gestor recuperable. En CICS para z/OS, no obstante, **SysLib.commit** da como resultado un SYNCPOINT (punto de sincronismo) de CICS, que realiza un compromiso de dos fases que se coordina entre todos los gestores de recursos.

**SysLib.commit** libera los bloqueos de posición de exploración y los de actualización en cualquier archivo o base de datos.

Cuando se utiliza **SysLib.commit** con registros MQ, se aplica lo siguiente:

- Las actualizaciones de colas de mensajes sólo son recuperables si la opción *Incluir mensaje en transacción* está seleccionada en el componente de registro MQ.
- Tanto las sentencias **get** como las sentencias **add** de mensajes se ven afectadas por las sentencias **commit** y **rollback** para mensajes recuperables. Si se emite una sentencia **rollback** después de una sentencia **get** para un mensaje recuperable, el mensaje se vuelve a colocar en la cola de entrada de modo que el mensaje de entrada no se pierde cuando la transacción no finaliza satisfactoriamente. Además, si se emite una sentencia **rollback** después de una sentencia **add** para un mensaje recuperable, el mensaje se suprime de la cola.

Puede mejorar el rendimiento evitando el uso innecesario de **SysLib.commit**. Para conocer detalles sobre cuándo se produce un compromiso implícito, consulte *Unidad lógica de trabajo*.

#### Ejemplo:

```
sysLib.commit();
```

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“Unidad lógica de trabajo” en la página 307

“Soporte de MQSeries” en la página 265

“Unidad de ejecución” en la página 742

“Soporte de SQL” en la página 229

#### Consulta relacionada

“commitOnConverse” en la página 922

“segmentedMode” en la página 925

“Biblioteca SysLib de EGL” en la página 887

“Propiedad luwControl del elemento callLink” en la página 414

“open” en la página 616

“prepare” en la página 630

#### conditionAsInt()

La función de sistema **SysLib.conditionAsInt** acepta una expresión lógica (como por ejemplo *myVar == 6*), devolviendo un 1 si la expresión es true y un 0 si la expresión es false.

```
SysLib.conditionAsInt(expresiónLógica AnyLogicalExpression in)
returns (resultado SMALLINT)
```

*resultado*

Un valor de tipo SMALLINT.

*expresiónLógica*

Una expresión lógica, tal como se describe en el apartado *Expresiones lógicas*.

#### Ejemplo:

```
myField = -5;

// resultado = 0
result = SysLib.conditionAsInt(myField == 6);
```

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

## Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

“Expresiones lógicas” en la página 497

## connect()

La función de sistema **SysLib.connect** permite que un programa se conecte a una base de datos durante la ejecución. Esta función no devuelve ningún valor.

```
SysLib.connect(
 baseDatos STRING in,
 IDusuario STRING in,
 contraseña STRING in
 [, ámbitoCompromiso enumerationCommitScope in
 [, opciónDesconexión enumerationDisconnectOption in
 [, nivelAislamiento enumerationIsolationLevel in
 [, controlCompromiso enumerationCommitControlOption in
]]])
```

*baseDatos*

Identifica una base de datos:

- Si *baseDatos* se establece en RESET se vuelve a conectar a la base de datos por omisión, pero si la base de datos por omisión no está disponible, el estado de conexión sigue igual; para obtener más información, consulte la sección *Base de datos por omisión*.
- De lo contrario, busque el nombre de la base de datos física consultando la propiedad **vgj.jdbc.database.servidor**, donde *servidor* es el nombre de la base de datos especificada en la llamada a **SysLib.connect**. Si esta propiedad no está definida, el nombre de la base de datos que está especificado en la llamada a **SysLib.connect** se utiliza tal cual.
- El formato del nombre de base de datos es distinto para las conexiones J2EE que para las conexiones no J2EE:
  - Si ha generado el programa para un entorno J2EE, utilice el nombre al que está enlazado el origen de datos en el registro JNDI; por ejemplo, jdbc/MyDB. Esta situación se produce si la opción del descriptor de construcción **J2EE** está establecida en YES.
  - Si ha generado el programa para un entorno JDBC no J2EE, utilice un URL de conexión; por ejemplo, jdbc:db2:MyDB. Esta situación se produce si la opción **J2EE** está establecida en NO.

*idUsuario*

ID de usuario que se utiliza para acceder a la base de datos. El argumento debe ser un elemento de tipo CHAR y longitud 8, y es válido un literal. El argumento es obligatorio. Para obtener información general, consulte la sección *Autorización de base de datos y nombres de tabla*.

*contraseña*

Contraseña que se utiliza para acceder a la base de datos. El argumento debe ser un elemento de tipo CHAR y longitud 8, y es válido un literal. El argumento es obligatorio.

*ámbitoCompromiso*

El valor es una de las siguientes palabras, y no pueden utilizarse comillas ni una variable:

### **type1 (el valor por omisión)**

Sólo se soporta un compromiso de *una* fase. Una nueva conexión cierra todos los cursores, libera los bloqueos y finaliza cualquier conexión existente; sin embargo, invoque **SysLib.commit** o **SysLib.rollback** antes de realizar una conexión *type1*.

Si utiliza `type1` como valor de *ámbitoCompromiso*, el valor del parámetro *opciónDesconexión* debe ser la palabra *explicit*, ya que es el valor por omisión.

### **type2**

Una conexión a una base de datos no cierra cursores, libera bloqueos ni finaliza una conexión existente. Aunque puede utilizar varias conexiones para leer varias bases de datos, sólo debe actualizar una base de datos en una unidad de trabajo ya que sólo está disponible un compromiso de una fase.

### **twophase**

Idéntico a `type2`.

### *opciónDesconexión*

Este parámetro sólo es significativo si se genera salida Java. El valor es una de las siguientes palabras, y no pueden utilizarse comillas ni una variable:

#### **explicit (el valor por omisión)**

La conexión permanece activa después de que el programa invoca **SysLib.commit** o **SysLib.rollback**. Para liberar los recursos de la conexión, un programa debe emitir `SysLib.disconnect`.

Si utiliza `type1` como valor de *ámbitoCompromiso*, el valor del parámetro *opciónDesconexión* debe establecerse en (o dejar que tome por omisión) la palabra *explicit*.

#### **automatic**

Una operación de compromiso o retrotracción finaliza una conexión existente.

#### **conditional**

Una operación de compromiso o retrotracción finaliza automáticamente una conexión existente, a menos que esté abierto un cursor y la opción de retención (*hold*) esté en vigor para ese cursor. Para obtener información detallada sobre la opción *hold*, consulte la sección *open*.

### *nivelAislamiento*

Este parámetro indica el nivel de independencia de una transacción de base de datos con respecto a otra

las palabras siguientes están por orden de severidad y, al igual que antes, no puede utilizar comillas ni una variable:

- **readUncommitted**
- **readCommitted**
- **repeatableRead**
- **serializableTransaction** (el valor por omisión)

Para obtener información detallada, consulte la documentación de JDBC de Sun Microsystems, Inc.

### *controlCompromiso*

Este parámetro especifica si se produce un compromiso después de cada cambio en la base de datos.

Los valores válidos son los siguientes:

- **noAutoCommit** (el valor por omisión) significa que el compromiso no es automático, lo que habitualmente implica una ejecución más rápida. Para conocer los detalles acerca de las reglas de compromiso y retrotracción en este caso, consulte la sección *Unidad de trabajo lógica*.

- **autoCommit** significa que las actualizaciones se producen inmediatamente.

Puede pasar de autoCommit a noAutoCommit temporalmente. Para obtener más detalles, consulte la sección *SysLib.beginDatabaseTransaction*.

**Consideraciones de definición:** **SysLib.connect** establece las siguientes variables de sistema:

- VgVar.sqlerrd[3]
- SysVar.sqlca
- SysVar.sqlcode
- VgVar.sqlWarn[2]

#### **Ejemplo:**

```
SysLib.connect(myDatabase, myUserid, myPassword);
```

#### **Conceptos relacionados**

“Unidad lógica de trabajo” en la página 307

“Unidad de ejecución” en la página 742

“Soporte de SQL” en la página 229

#### **Tareas relacionadas**

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

“Establecer una conexión JDBC J2EE” en la página 362

“Cómo se realiza una conexión JDBC estándar” en la página 263

#### **Consulta relacionada**

“Autorización de base de datos y nombres de tabla” en la página 466

“Base de datos por omisión” en la página 251

“Biblioteca SysLib de EGL” en la página 887

“Propiedades de ejecución de Java (detalles)” en la página 540

“open” en la página 616

“sqlDB” en la página 400

“beginDatabaseTransaction()” en la página 889

“disconnect()” en la página 900

“sqlca” en la página 935

“sqlcode” en la página 936

“sqlerrd” en la página 949

“sqlerrmc” en la página 950

“sqlWarn” en la página 951

#### **convert()**

La función de sistema **SysLib.convert** convierte datos entre los formatos EBCDIC (sistema principal) y ASCII (estación de trabajo) o bien realiza la conversión de página de códigos dentro de un único formato. Puede utilizar **SysLib.convert** como nombre de función en una sentencia de invocación de función.

```
SysLib.convert(
 destino anyFixedItemOrRecordOrFormVariable inout,
 dirección enumerationConversionDirection in,
 tablaConversión CHAR(8) in)
```

*destino*

Nombre del registro, elemento de datos o formulario que tiene el formato que se desea convertir. Los datos se convierten en su sitio en función de la definición de elemento de los elementos de nivel más bajo (elementos sin subestructura) en el objeto destino.

Los registros de longitud variable sólo se convierten para la longitud del registro actual. La longitud del registro actual se calcula utilizando el elemento `numElementsItem` del registro o se establece a partir del elemento `lengthItem` del registro. Se produce un error de conversión y el programa finaliza si el registro de longitud variable finaliza en medio de un campo numérico o un carácter DBCHAR.

#### *dirección*

Dirección de la conversión. "R" y "L" (incluidas las comillas) son los únicos valores válidos. Es obligatoria si se especifica *tablaConversión*; en caso contrario, es opcional.

"R" Valor por omisión. Se supone que los datos están en formato remoto y se convierten a formato local.

"L" Se supone que los datos están en formato local y se convierten a formato remoto (tal como se define en la tabla de conversión).

#### *tablaConversión*

Elemento de datos o literal (ocho caracteres, opcional) que especifica el nombre de la tabla de conversión que se utilizará para la conversión de datos. El valor por omisión es la tabla de conversión asociada al código de idioma nacional especificado cuando se generó el programa.

**Consideraciones de definición:** Puede utilizar el componente de opciones de enlace para solicitar que se genere la conversión de datos automática para llamadas remotas, para iniciar transacciones asíncronas remotas o para el acceso a archivos remotos. La conversión automática se realiza siempre utilizando la estructura de datos definida para el argumento que se está convirtiendo. Si un argumento tiene varios formatos, no solicite la conversión automática. En su lugar, codifique el programa para llamar explícitamente a **SysLib.convert** con las declaraciones de registro redefinidas que se correlacionan correctamente con los valores actuales del argumento.

#### **Ejemplo:**

```
Record RecordA
 record_type char(3);
 item1 char(20);
end

Record RecordB
 record_type char(3);
 item2 bigint;
 item3 decimal(7);
 item4 char(8);
end

Program ProgramX type basicProgram
 myRecordA RecordA;
 myRecordB RecordB {redefines = "myRecordA"};
 myConvTable char(8);

 function main();
 myConvTable = "ELACNENU"; // conversion table for US English
 if (myRecordA.record_type == "00A")
 SysLib.convert(myRecordA, "L", myConvTable);
 else;
 SysLib.convert(myRecordB, "L", myConvTable);
 end
 call ProgramY myRecordA;
 end
end
```

## Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

## Consulta relacionada

“Conversión de datos” en la página 467

“Biblioteca SysLib de EGL” en la página 887

“callConversionTable” en la página 930

## defineDatabaseAlias()

La función del sistema **SysLib.defineDatabaseAlias** crea un alias que puede utilizarse para establecer una conexión nueva con una base de datos a la que el código ya está conectado. Una vez establecido, el alias puede utilizarse en cualquiera de estas funciones:

- SysLib.connect
- SysLib.disconnect
- SysLib.beginDatabaseTransaction
- SysLib.setCurrentDatabase
- VGLib.connectionService

El alias también puede utilizarse en el campo **connectionName** de una variable de tipo **ReportData**.

```
SysLib.defineDatabaseAlias(
 alias STRING in,
 baseDatos STRING in)
```

### *alias*

Un literal de serie o una variable que actúa como alias de la conexión identificada en el segundo parámetro. El alias no es sensible a las mayúsculas/minúsculas.

### *base de datos*

Un nombre de base de datos especificado en SysLib.connect o VGLib.connectionService. Utilice un literal o variable de un tipo de carácter.

Si no especifica una conexión, la función afecta a la conexión actual.

## Ejemplos:

```
// Conectar con una base de datos con el alias "alias",
// que se convierte en la conexión actual.
defineDatabaseAlias("alias", "database");
connect("alias", "user", "pwd");

// Hacer dos conexiones con la misma base de datos.
String db = "database";
defineDatabaseAlias("alias1", db);
defineDatabaseAlias("alias2", db);
connect("alias1", "user", "pwd");
connect("alias2", "user", "pwd");

// Otra forma de hacer dos conexiones
// con la misma base de datos.
defineDatabaseAlias("alias", "database");
connect("alias", "user", "pwd");
connect("database", "user", "pwd");

// Una alias definido pero no usado. El segundo
// connect() no crea una conexión nueva.
defineDatabaseAlias("alias", "database");
connect("database", "user", "pwd");
connect("database", "user", "pwd");
```

```

// Utilización de alias (no sensible a mayús./minús.)
// al desconectar.
defineDatabaseAlias("alias", "database");
connect("aLiAs", "user", "pwd");
disconnect("ALIAS");

// La llamada a disconnect siguiente falla porque
// la conexión se llama "alias" no "database".
defineDatabaseAlias("alias", "database");
connect("alias", "user", "pwd");
disconnect("database");

// Un alias puede cambiar. Después de la llamada siguiente
// "alias" hace referencia a "firstDatabase"
defineDatabaseAlias("alias", "firstDatabase");

// Después de la llamada siguiente
// "alias" hace referencia a "secondDatabase".
defineDatabaseAlias("alias", "secondDatabase");

// La última llamada habría fallado
// si hubiera una conexión con "alias".

```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“Soporte de SQL” en la página 229

### Consulta relacionada

“beginDatabaseTransaction()” en la página 889 “connect()” en la página 895

“disconnect()”

“setCurrentDatabase()” en la página 906

“connectionService()” en la página 916

## disconnect()

La función de sistema **SysLib.disconnect** efectúa la desconexión de la base de datos especificada o (si no se ha especificado ninguna) de la base de datos actual.

```

SysLib.disconnect(
 [baseDatos STRING in
]
)

```

*baseDatos*

Un nombre de base de datos especificado en SysLib.connect o VGLib.connectionService. Utilice un literal o variable de un tipo de carácter.

Antes de realizar la desconexión, invoque SysLib.commit o SysLib.rollback.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

“commit()” en la página 893

“connect()” en la página 895

“rollback()” en la página 905

“connectionService()” en la página 916

## disconnectAll()

La función de sistema **SysLib.disconnectAll** efectúa la desconexión de todas las bases de datos conectadas actualmente.

Antes de realizar la desconexión, invoque **SysLib.commit** o **SysLib.rollback**.

**SysLib.disconnectAll**( )

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

“connect()” en la página 895

“connectionService()” en la página 916

## errorLog()

La función de sistema **SysLib.errorLog** copia texto en las anotaciones de error que ha iniciado la función de sistema **SysLib.startLog**.

**SysLib.errorLog**(*texto* **STRING** *in*)

*texto*

El valor que debe colocarse en las anotaciones de error.

Las entradas en las anotaciones incluyen la fecha y la hora en que se ha escrito la entrada.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

“startLog()” en la página 911

## getCmdLineArg()

La función del sistema **SysLib.getCmdLineArg** devuelve el argumento especificado de la lista de argumentos con los que se invocó el programa EGL. El argumento especificado se devuelve como un valor de serie.

**SysLib.getCmdLineArg**(*índice* **INT** *in*)  
returns (*resultado* **STRING**)

*resultado*

El *resultado* puede ser cualquier elemento de caracteres.

*índice*

El *índice* puede ser cualquier elemento entero.

- Si *índice* = 0, se devuelve el nombre del mandato.
- Si *índice* = *n*, se devuelve el *n*ésimo nombre de argumento.
- Si *n* es mayor que la cuenta de argumentos, se devuelve un espacio en blanco.

El ejemplo de código siguiente recorre circularmente la lista de argumentos:

```
count int;
argument char(20);

count = 0;
argumentCount = SysLib.getCmdLineArgCount();
```

```

while (count < argumentCount)
 argument = SysLib.getCmdLineArg(count)
 count = count + 1;
end

```

La función **SysLib.getCmdLineArg** sólo está soportada en entornos Java.

Utilice la función **SysLib.getCmdLineArgCount** para obtener el número de argumentos o parámetros que se pasaron al programa EGL principal en el momento de su invocación.

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

#### Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

“getCmdLineArgCount()”

### getCmdLineArgCount()

La función del sistema **SysLib.getCmdLineArgCount** devuelve el número de argumentos que se han utilizado para iniciar el programa EGL principal.

```

SysLib.getCmdLineArgCount()
returns (resultado INT)

```

*resultado*

El *resultado* es el número de argumentos.

El ejemplo de código siguiente recorre circularmente la lista de argumentos:

```

count int;
argument char(20);

count = 0;
argumentCount = SysLib.getCmdLineArgCount();

while (count < argumentCount)
 argument = SysLib.getCmdLineArg(count)
 count = count + 1;
end

```

La función **SysLib.getCmdLineArgCount** sólo está soportada en entornos Java.

Utilice la función **SysLib.getCmdLineArg** para obtener el argumento especificado de la lista de argumentos con los que se ha invocado el programa EGL.

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

#### Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

“getCmdLineArg()” en la página 901

### getMessage()

La función del sistema **SysLib.getMessage** devuelve un mensaje del archivo al que se hace referencia en la propiedad de tiempo de ejecución Java `vgj.message.file`. Puede especificar inserciones para incluirlas en el mensaje. Después de recibir el mensaje, puede visualizarlo en un formulario de texto, un formulario de impresión, un formulario de consola, una página Web o un archivo de anotaciones.

```
SysLib.getMessage(
 clave STRING in
 [, insertArray STRING[] in])
returns (resultado STRING)
```

*resultado*

Un campo de tipo **STRING**.

*clave*

Un campo de caracteres o un literal de tipo **STRING**. Este parámetro proporciona la clave en el archivo de propiedades que se utiliza en tiempo de ejecución. Si la clave está en blanco, el mensaje es una concatenación de inserciones de mensaje.

*matrizInserción*

Una matriz de tipo **STRING**. Cada elemento contiene una inserción para incluirla en el mensaje que se recupera.

En el texto del mensaje, el símbolo de sustitución es un entero entre corchetes, como en este ejemplo de un archivo de propiedades:

```
VGJ0216E = {0} no es una
máscara de fecha válida para {1}.
```

El primer elemento de *matrizInserción* se asigna al marcador de posición con el número uno, etc.

El formato del archivo al que hace referencia la propiedad de tiempo de ejecución Java property `vgj.messages.file` es el mismo que para cualquier archivo de propiedades Java. Para obtener detalles sobre ese formato, consulte la sección *Archivo de propiedades de programa*.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“Propiedades de tiempo de ejecución Java” en la página 347

“Archivo de propiedades del programa” en la página 350

### Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

“Propiedades de ejecución de Java (detalles)” en la página 540

## getProperty()

La función del sistema **SysLib.getProperty** recupera el valor de una propiedad de tiempo de ejecución Java. Si no se encuentra la propiedad especificada, la función devuelve una serie nula (“”).

```
SysLib.getProperty(nombrePropiedad STRING in)
returns (resultado STRING)
```

*resultado*

Un campo de tipo **STRING**

*nombrePropiedad*

Una constante o variable de caracteres o un literal de serie

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“Propiedades de tiempo de ejecución Java” en la página 347

### Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

“Propiedades de ejecución de Java (detalles)” en la página 540

## loadTable()

La función del sistema **SysLib.loadTable** carga información de un archivo en una base de datos relacional.

```
SysLib.loadTable(
 nombreArchivo STRING in,
 insertarEnCláusula STRING in
 [, delimitador STRING in
])
```

*nombreArchivo*

El nombre del archivo. El nombre está totalmente calificado o es relativo al directorio desde el que se invoca el programa.

*insertarEnCláusula*

Especifique la tabla y las filas que suministrarán los datos. Utilice la sintaxis de una cláusula INSERT de una sentencia SQL INSERT, como en este ejemplo:

```
"INSERT INTO miTabla(columna1, columna2)"
```

Una cláusula como la siguiente es suficiente si el archivo incluye valores para todas las columnas de tabla en orden de columnas:

```
"INSERT INTO miTabla"
```

*delimitador*

Especifica el símbolo utilizado para separar un valor del siguiente en el archivo. (Una fila de datos debe estar separada de la siguiente mediante el carácter de salto de línea).

El símbolo por omisión para el *delimitador* es el valor de la propiedad de entorno de ejecución Java **vgj.default.databaseDelimiter**; y el valor por omisión de esa propiedad es la barra vertical (|).

Los símbolos siguientes no están disponibles:

- Caracteres hexadecimales (0 a 9, a a f, A a F)
- Barra inclinada invertida (\)
- El carácter de salto de línea o *CONTROL-J*

Para descargar información de una tabla de base de datos relacional e insertarla en un archivo, utilice la función **SysLib.unloadTable**.

### Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

“Propiedades de ejecución de Java (detalles)” en la página 540

“unloadTable()” en la página 912

## maximumSize()

La función de sistema **SysLib.maximumSize** devuelve el número máximo de filas que puede haber en una matriz dinámica; específicamente, la función devuelve el valor de la propiedad de matriz **maxSize**.

```
SysLib.maximumSize(nombreMatriz anyArray in)
returns (resultado INT)
```

*resultado*

Número máximo de filas.

*nombreMatriz*

Nombre de la matriz dinámica.

**Consideraciones acerca de la definición:** El elemento al que se devuelve el valor de ser de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

El nombre de matriz puede calificarse mediante un nombre de paquete, un nombre de biblioteca o ambos

Si se hace referencia a un elemento o registro que no es una matriz dinámica, se produce un error.

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Matrices” en la página 74

“Biblioteca SysLib de EGL” en la página 887

### **queryCurrentDatabase()**

La función de sistema **SysLib.queryCurrentDatabase** devuelve el producto y el número de release de la base de datos conectada actualmente.

```
SysLib.queryCurrentDatabase(
 producto CHAR(8) inOut,
 release CHAR(8) inOut)
```

#### *producto*

Recibe el nombre de producto de base de datos. El argumento debe ser un elemento de tipo CHAR y longitud 8.

Para determinar la serie que se recibirá cuando el código se conecte a una base de datos determinada, consulte la documentación del producto de base de datos o controlador, o ejecute el código en un entorno de prueba y escriba el valor recibido en un archivo.

#### *release*

Recibe el nivel de release de la base de datos. El argumento debe ser un elemento de tipo CHAR y longitud 8.

Para determinar la serie que se recibirá cuando el código se conecte a una base de datos determinada, consulte la documentación del producto de base de datos o controlador, o ejecute el código en un entorno de prueba y escriba el valor recibido en un archivo.

#### **Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

#### **Consulta relacionada**

“Biblioteca SysLib de EGL” en la página 887

### **rollback()**

La función de sistema **SysLib.rollback** invierte las actualizaciones efectuadas en bases de datos y colas de mensajes MQSeries desde la última operación de compromiso. Esta inversión se produce en cualquier aplicación generada por EGL.

```
SysLib.rollback()
```

Se produce automáticamente una retrotracción cuando un programa finaliza debido a una condición de error.

**Consideraciones de definición:** Cuando se utiliza **SysLib.rollback** con registros MQ, se aplica lo siguiente:

- Las actualizaciones de colas de mensajes sólo son recuperables si la opción *Incluir mensaje en transacción* está seleccionada en el componente de registro MQ.
- Tanto las sentencias **scan** como las sentencias **add** de mensajes se ven afectadas por las sentencias **commit** y **rollback** para mensajes recuperables. Si se emite una sentencia **rollback** después de una sentencia **scan** para un mensaje recuperable, el mensaje se vuelve a colocar en la cola de entrada de modo que el mensaje de entrada no se pierde cuando la transacción no finaliza satisfactoriamente. Además, si se emite una sentencia **rollback** después de una sentencia **add** para un mensaje recuperable, el mensaje se suprime de la cola.

**Plataformas destino:**

Plataforma	Consideraciones sobre compatibilidad
iSeries, USS, Windows 2000, Windows NT	Invierte los cambios realizados en bases de datos relacionales y colas de mensajes MQSeries, así como los cambios efectuados en programas de servidor remoto llamados mediante una unidad de trabajo controlada por el cliente.

**Ejemplo:**

```
SysLib.rollback();
```

**Conceptos relacionados**

- “Diagrama de sintaxis para funciones EGL” en la página 754
- “Unidad lógica de trabajo” en la página 307
- “Soporte de MQSeries” en la página 265
- “Soporte de SQL” en la página 229

**Consulta relacionada**

“Biblioteca SysLib de EGL” en la página 887

**setCurrentDatabase()**

La función de sistema **SysLib.setCurrentDatabase** convierte la base de datos especificada en la que está activa actualmente.

```
SysLib.setCurrentDatabase(baseDatos STRING in)
```

*baseDatos*

Un nombre de base de datos especificado en SysLib.connect o VGLib.connectionService. Utilice un literal o variable de un tipo de carácter.

**Conceptos relacionados**

“Diagrama de sintaxis para funciones EGL” en la página 754

**Consulta relacionada**

- “Biblioteca SysLib de EGL” en la página 887
- “connect()” en la página 895
- “connectionService()” en la página 916

**setError()**

La función de sistema **SysLib.setError** asocia un mensaje con un elemento de un PageHandler o con la totalidad del PageHandler . El mensaje se coloca en la ubicación de un mensaje o código de mensajes JSF del JSP y se visualiza cuando lo hace la página Web relacionada.

Si una función de validación invoca **SysLib.setError**, la página Web vuelve a visualizarse automáticamente cuando finaliza la función.

```
SysLib.setError(
 elementoEnError anyPageItem in,
 claveMsj STRING in
 {, elementoInserción sysLibItemInsert in})
```

```
SysLib.setError(
 this enumerationThis in,
 claveMsj STRING in
 {, elementoInserción sysLibItemInsert in})
```

```
SysLib.setError(textoMsj STRING in)
```

*elementoEnError*

El nombre del elemento de PageHandler que contiene el error.

**this**

Hace referencia al PageHandler desde el que se emite **SysLib.setError**. En este caso, el mensaje no es específico de un elemento, sino que está asociado con la totalidad del PageHandler. Para obtener detalles acerca de **this**, consulte el apartado *Referencias a variables y constantes*.

*claveMsj*

Un elemento o literal de carácter (tipo CHAR o MBCHAR) que proporciona la clave en el empaquetamiento de recursos de mensaje o archivo de propiedades utilizado durante la ejecución. Si la clave está en blanco, el mensaje es una concatenación de inserciones de mensaje.

*elementoInserción*

El elemento o literal de carácter incluido como inserción en el mensaje de salida. El símbolo de sustitución del texto de mensaje es un entero entre corchetes, como el de este ejemplo:

```
Nombre de archivo no válido {0}
```

*textoMsj*

El elemento o literal de carácter que puede especificar si no especifica otros argumentos. El texto se asocia a la totalidad de la página.

Puede asociar varios elementos con un elemento o PageHandler. El entorno de ejecución EGL visualiza los mensajes cuando se vuelve a visualizar la página. Si el control se reenvía (concretamente, si el PageHandler ejecuta una sentencia **forward**), estos mensajes se pierden.

### Conceptos relacionados

“PageHandler” en la página 194

“Referencias a variables en EGL” en la página 58

### Tareas relacionadas

“Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755

### Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

“forward” en la página 583

## setLocale()

La función de sistema **SysLib.setLocale** se utiliza en los PageHandlers. La función establece el entorno local Java, que determina los siguientes aspectos del comportamiento de ejecución:

- El idioma utilizado para etiquetas y mensajes

- Los formatos de fecha y hora por omisión

Puede presentar una lista de idiomas en una página Web, por ejemplo, y establecer el entorno local Java en función de la selección del usuario. El nuevo entorno local Java se utilizará hasta que se produzca una de las siguientes situaciones:

- Se invoca de nuevo **SysLib.setLocale**; o bien
- Finaliza la sesión del navegador; o
- Se presenta una nueva página Web.

En los casos mencionados, la próxima página Web volverá (por omisión) al entorno local Java especificado en el navegador.

Si el usuario somete un formulario o pulsa un enlace que abre una ventana nueva, el entorno local Java de la ventana original no resulta afectado por el entorno local de la ventana nueva.

**SysLib.setLocale** se ajusta a la documentación de la API del JDK 1.1 y 1.2 para la clase `java.util.Locale`. Consulte el ISO 639 para conocer los códigos de idioma y el ISO 3166 para conocer los códigos de país.

```
SysLib.setLocale(
 códigoIdioma CHAR(2) in,
 códigoPaís CHAR(2) in
 [, variante CHAR(2) in])
```

*códigoIdioma*

Un código de idioma de dos caracteres especificado como literal o contenido en un elemento de tipo CHAR. Sólo son válidos los códigos de idioma definidos en el ISO 639.

*códigoPaís*

Un código de país de dos caracteres especificado como literal o contenido en un elemento de tipo CHAR. Sólo son válidos los códigos de país definidos en el ISO 3166.

*variante*

Una variante, que es un código especificado como literal o contenido en un elemento de tipo CHAR. Este código no forma parte de una especificación Java, sino que depende del navegador y de otros aspectos del entorno del usuario.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

“PageHandler” en la página 194

### Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

## setRemoteUser()

La función de sistema **SysLib.setRemoteUser** establece el ID de usuario y la contraseña utilizados en las llamadas a programas remotos.

```
SysLib.setRemoteUser(
 IDusuario STRING in,
 contraseña STRING in)
```

*IDusuario*

El ID de usuario en el sistema remoto.

*contraseña*

La contraseña en el sistema remoto.

Cuando el componente de opción de enlace, elemento callLink, propiedad remoteComType es CICSJ2C, CICSECI o JAVA400 en una llamada remota, la autorización se basa en los valores (si no están en blanco) que se pasan a **SysLib.setRemoteUser**. Si un valor está en blanco o no se especifica, el valor se busca en el archivo **csouidpwd.properties**, que incluye las propiedades CSOUID (para el ID de usuario) y CSOPWD (para la contraseña). Si no se utiliza ninguno de los métodos, el entorno de ejecución de EGL realiza la llamada sin nombre de usuario ni contraseña.

Antes de invocar **SysLib.setRemoteUser**, el código puede emitir funciones de acceso a Java que visualicen un recuadro de diálogo para solicitar el usuario el ID de usuario y la contraseña. Puede utilizar uno o los dos valores de **csouidpwd.properties** como valor por omisión, que entrará en vigor cuando el usuario no suministre la información.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Archivo csouidpwd.properties para llamadas remotas” en la página 378

“Biblioteca SysLib de EGL” en la página 887

“Propiedad remoteComType del elemento callLink” en la página 419

### size()

La función de sistema **SysLib.size** devuelve el número de filas de la tabla de datos especificada o el número de elementos de la matriz especificada. La matriz puede ser una matriz de elementos de estructura o una matriz dinámica de elementos de datos o registros.

```
SysLib.size(nombreMatriz anyArray in)
returns (resultado INT)
```

*resultado*

El número de filas de la tabla de datos especificada o el número de elementos de la matriz especificada.

*nombreMatriz*

Nombre de la matriz o tabla de datos.

**Consideraciones acerca de la definición:** El elemento al que se devuelve el valor de ser de tipo INT o el siguiente equivalente: tipo BIN con longitud 9 y sin posiciones decimales.

Si el nombre de matriz (*nombreMatriz*) está en un elemento subestructurado de otra matriz, el valor devuelto es el número de apariciones del propio elemento de estructura, no el número total de apariciones de la estructura que lo contiene (consulte la sección *Ejemplos*).

El nombre de matriz puede calificarse mediante un nombre de paquete, un nombre de biblioteca o ambos

Si se hace referencia a un elemento o registro que no es una matriz, se produce un error.

**Ejemplos:** Este ejemplo utiliza el valor devuelto por **SysLib.size** para controlar un bucle:

```
// Calcular la suma de una matriz de números
sum = 0;
i = 1;
myArraySize = SysLib.size(myArray);

while (i <= myArraySize)
 sum = myArray[i] + sum;
 i = i + 1;
end
```

A continuación, considere el siguiente componente de registro:

```
Record myRecordPart
 10 siTop CHAR(40)[3];
 20 siNext CHAR(20)[2];
end
```

Dado que ha creado un registro basado en **myRecordPart**, puede utilizar **SysLib.size(siNext)** para determinar el valor de apariciones (occurs) de la matriz subordinada:

```
// Establece count en 2
count = SysLib.size(myRecord.siTop.siNext);
```

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Matrices” en la página 74

“Biblioteca SysLib de EGL” en la página 887

## startCmd()

La función del sistema **SysLib.startCmd** ejecuta un mandato del sistema y no espera a que finalice el mandato.

```
SysLib.startCmd(
 serieMandato STRING in
 [, serieModalidad STRING in
])
```

*serieMandato*

Identifica el mandato del sistema operativo que se desea invocar.

*serieModalidad*

La *serieModalidad* puede ser cualquier carácter o elemento de serie. El elemento puede estar en cualquiera de las dos modalidades:

- *formulario*: en la que cualquiera carácter de entrada queda a disposición del programa una vez teclado, es decir, cada pulsación de tecla se pasa directamente al mandato especificado.
- *línea*: en la que la entrada no está disponible hasta que se utiliza el carácter de línea nueva, es decir, no se envía ninguna información al mandato especificado hasta que se pulsa la tecla INTRO y entonces se envía toda la línea al mandato.

El ejemplo de código siguiente

```
ejemplo de Arlan aquí...
```

El mandato del sistema que se está ejecutando debe ser visible para el programa en ejecución. Por ejemplo, si ejecuta **callCmd("mySpecialProgram.exe")**, el programa

"mySpecialProgram.exe" debe estar en un directorio incluido en la variable de entorno PATH. También puede especificar la ubicación completa del directorio, por ejemplo `callCmd("program files/myWork/mySpecialProgram.exe")`.

La función `SysLib.startCmd` sólo está soportada en entornos Java.

Utilice la función `SysLib.callCmd` para ejecutar un mandato del sistema que espera a que finalice el mandato.

#### Conceptos relacionados

"Diagrama de sintaxis para funciones EGL" en la página 754

#### Consulta relacionada

"Biblioteca SysLib de EGL" en la página 887

Cambio falso según sea necesario

### startLog()

La función de sistema `SysLib.startLog` abre un archivo de anotaciones de error. El texto se escribe en dichas anotaciones cada vez que el programa invoca `SysLib.errorLog`.

```
SysLib.startLog(archivoAnotaciones STRING in)
```

*archivoAnotaciones*

El archivo de anotaciones de error.

#### Conceptos relacionados

"Diagrama de sintaxis para funciones EGL" en la página 754

#### Consulta relacionada

"Biblioteca SysLib de EGL" en la página 887

"errorLog()" en la página 901

### startTransaction()

La función de sistema `SysLib.startTransaction` invoca de forma asíncrona un programa principal, asocia ese programa con un dispositivo de impresora o un terminal y pasa un registro. Si EGL genera el programa receptor, el registro se utiliza para inicializar el registro de entrada; si VisualAge Generator produce el receptor, el registro se utiliza para inicializar el almacenamiento de trabajo.

El comportamiento por omisión de esta función consiste en iniciar un programa que reside en el mismo paquete de Java. Para cambiar ese comportamiento, especifique un elemento `asynchLink` en el componente de opciones de enlace utilizado para generar el programa de invocación.

Un programa Java sólo puede efectuar la transferencia a otro programa Java de la misma máquina.

```
SysLib.startTransaction(
 petición anyBasicRecord in
 [, idImpr startTransactionPrId in
 [, idTerm CHAR(4) in]])
```

*petición*

El nombre de un registro básico, que debe tener el siguiente formato:

- Los 2 primeros bytes (de tipo SMALLINT o de tipo BIN sin decimales) contienen la longitud de los datos que deben pasarse a la transacción iniciada, más 10 para los dos campos (incluido este) que no se pasan.

- Los 8 bytes siguientes (de tipo CHAR) tampoco se pasan, pero contienen el nombre del programa que debe iniciarse.
- El resto del registro de petición se pasa.

#### *idImpr*

Si se especifica este elemento de 4 bytes, se ignorará.

#### *idTerm*

Si se especifica este elemento de 4 bytes de tipo CHAR, se ignorará. Debe especificar *prID* si especifica *termID*.

### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

### Consulta relacionada

“Elemento asynchLink” en la página 377

“Biblioteca SysLib de EGL” en la página 887

“errorCode” en la página 931

“printerAssociation” en la página 923

“transfer” en la página 646

## unloadTable()

La función del sistema **SysLib.unloadTable** descarga información de una tabla de base de datos relacional en un archivo.

```
SysLib.unloadTable(
 nombreArchivo STRING in,
 sentenciaSelect STRING in
 [, delimitador STRING in
])
```

#### *nombreArchivo*

El nombre del archivo. El nombre está totalmente calificado o es relativo al directorio desde el que se invoca el programa.

#### *sentenciaSelect*

Especifique los criterios de selección de datos de la base de datos relacional. Utilice la sintaxis de una sentencia SQL SELECT sin incluir variables de lenguaje principal; por ejemplo:

```
"SELECT column1, column2 FROM myTABLE
WHERE column3 > 10"
```

#### *delimitador*

Especifica el símbolo utilizado para separar un valor del siguiente en el archivo. (Una fila de datos debe estar separada de la siguiente mediante el carácter de salto de línea).

El símbolo por omisión para el *delimitador* es el valor de la propiedad de entorno de ejecución Java **vgj.default.databaseDelimiter**; y el valor por omisión de esa propiedad es la barra vertical (|).

Los símbolos siguientes no están disponibles:

- Caracteres hexadecimales (0 a 9, a a f, A a F)
- Barra inclinada invertida (\)
- El carácter de salto de línea o *CONTROL-J*

Para cargar información de un archivo e insertarla en una tabla de base de datos relacional, utilice la función **SysLib.loadTable**.

### Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

“loadTable()” en la página 904

“Propiedades de ejecución de Java (detalles)” en la página 540

### verifyChkDigitMod10()

La función de sistema **SysLib.verifyChkDigitMod10** verifica un dígito de comprobación Modulus-10 en un elemento de carácter que empieza por una serie de enteros.

```
SysLib.verifyChkDigitMod10(
 texto anyChar in,
 longitudComprobación SMALLINT in,
 resultado SMALLINT inOut)
```

#### texto

Un elemento de carácter que empieza por una serie de enteros. El elemento incluye una posición adicional para el dígito de comprobación, situado inmediatamente a la derecha de los demás enteros.

#### longitudComprobación

Un elemento que contiene el número de caracteres que desea utilizar del elemento *texto*, incluida la posición utilizada para el dígito de comprobación. Este elemento tiene 4 dígitos y es de tipo SMALLINT o BIN, sin posiciones decimales.

#### resultado

Un elemento que recibe un de dos valores:

- 0, si el dígito de comprobación calculado coincide con el valor de *texto*
- 1, si el dígito de comprobación calculado no coincide con el valor

Este elemento tiene 4 dígitos y es de tipo SMALLINT o BIN, sin posiciones decimales.

Puede utilizar **SysLib.verifyChkDigitMod10** en una sentencia de invocación de función o como validador de elementos de un formulario de texto.

**Ejemplo:** En el ejemplo siguiente, *myInput* es un elemento de tipo CHAR y contiene el valor 1734284; *myLength* es un elemento de tipo SMALLINT y contiene el valor 7; y *myResult* es un elemento de tipo SMALLINT:

```
SysLib.verifyChkDigitMod10 (myInput, myLength, myResult);
```

Se utiliza un algoritmo para derivar el dígito de comprobación Modulus-10 y en ningún caso se tiene en cuenta el número de la posición del dígito de comprobación; pero, cuando el algoritmo se ha completado, el valor calculado se compara con el número de la posición del dígito de comprobación.

El algoritmo se describe en relación a los valores de ejemplo:

1. Multiplique la posición de unidades del número de entrada por 2 y multiplique todas las posiciones alternativas, de derecha a izquierda, por 2:

$$8 \times 2 = 16$$

$$4 \times 2 = 8$$

$$7 \times 2 = 14$$

2. Añada los dígitos de los productos (16814) a los dígitos del número de entrada (132) que no se hayan multiplicado por 2:

$$1 + 6 + 8 + 1 + 4 + 1 + 3 + 2 = 26$$

3. Para obtener el dígito de comprobación, sustraiga la suma del siguiente número más alto que termine en 0:

30 - 26 = 4

Si la sustracción produce 10, el dígito de comprobación será 0.

En este ejemplo, el dígito de comprobación calculado coincide con el valor de la posición del dígito de comprobación, y el valor de myResult es 0.

#### Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

“Propiedades de validación” en la página 67

### verifyChkDigitMod11()

La función de sistema **SysLib.verifyChkDigitMod11** verifica un dígito de comprobación Modulus-11 en un elemento de carácter que empieza por una serie de enteros.

```
SysLib.verifyChkDigitMod11(
 texto anyChar in,
 longitudComprobación SMALLINT in,
 resultado SMALLINT inOut)
```

#### texto

Un elemento de carácter que empieza por una serie de enteros. El elemento incluye una posición adicional para el dígito de comprobación, situado inmediatamente a la derecha de los demás enteros.

#### longitudComprobación

Un elemento que contiene el número de caracteres que desea utilizar del elemento *texto*, incluida la posición utilizada para el dígito de comprobación. Este elemento tiene 4 dígitos y es de tipo SMALLINT o BIN, sin posiciones decimales.

#### resultado

Un elemento que recibe un de dos valores:

- 0, si el dígito de comprobación calculado coincide con el valor de *texto*
- 1, si el dígito de comprobación calculado no coincide con el valor

Este elemento tiene 4 dígitos y es de tipo SMALLINT o BIN, sin posiciones decimales.

Puede utilizar **SysLib.verifyChkDigitMod11** en una sentencia de invocación de función o como validador de elementos de un formulario de texto.

**Ejemplo:** En el ejemplo siguiente, myInput es un elemento de tipo CHAR y contiene el valor 56621869; myLength es un elemento de tipo SMALLINT y contiene el valor 8; y myResult es un elemento de tipo SMALLINT:

```
sysLib.verifyChkDigitMod11 (myInput, myLength, myResult);
```

Se utiliza un algoritmo para derivar el dígito de comprobación Modulus-11 y en ningún caso se tiene en cuenta el número de la posición del dígito de comprobación; pero, cuando el algoritmo se ha completado, el valor calculado se compara con el número de la posición del dígito de comprobación. El algoritmo se describe en relación a los valores de ejemplo:

1. Multiplique el dígito de la posición de unidades del número de entrada por 2, el de la posición de decenas por 3, el de la posición de centenas por 4, y así sucesivamente, pero deje que myLength " 1 sea el número mayor utilizado como multiplicador; y, si el número de entrada contiene más dígitos, inicie de nuevo la secuencia utilizando 2 como multiplicador:

6 x 2 = 12  
8 x 3 = 24  
1 x 4 = 4  
2 x 5 = 10  
6 x 6 = 36  
6 x 7 = 42  
5 x 2 = 10

2. Sume los productos del primer paso y divida la suma por 11:

$(12 + 24 + 4 + 10 + 36 + 42 + 10) / 11$   
= 138 / 11  
= 12 resto 6

3. Para obtener el dígito de comprobación, sustraiga el resto de 11 para obtener el dígito de autocombprobación:

$11 - 6 = 5$

Si el resto es 0 o 1, el dígito de comprobación será 0.

En este ejemplo, el dígito de comprobación calculado coincide con el valor de la posición del dígito de comprobación, y el valor de myResult es 0.

#### Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

“Propiedades de validación” en la página 67

## wait()

La función de sistema **SysLib.wait** suspende la ejecución durante el número de segundos especificado.

**SysLib.wait**(*tiempoEnSegundos* **BIN(9,2)** *in*)

*tiempoEnSegundos*

El tiempo puede ser cualquier elemento o literal numérico. Si el número no es un entero, se llega hasta las centésimas de segundo.

Puede utilizar **SysLib.wait** cuando dos programas ejecutados asincrónicamente deben comunicarse a través de un registro de un archivo o base de datos compartidos. Puede que un programa necesite suspender el proceso hasta que el otro programa haya actualizado la información del registro compartido.

#### Ejemplo

```
SysLib.wait(15); // espera durante 15 segundos
```

#### Conceptos relacionados

“Diagrama de sintaxis para funciones EGL” en la página 754

#### Consulta relacionada

“Biblioteca SysLib de EGL” en la página 887

## Biblioteca VGLib de EGL

A continuación se muestran las funciones de VGLib:

Función de sistema/invocación	Descripción
<code>connectionService(idUsuario, contraseña, nombreServidor [, producto, release [, opciónConexión]])</code>	Proporciona dos ventajas: <ul style="list-style-type: none"><li>• Permite que un programa se conecte o desconecte a una base de datos durante la ejecución.</li><li>• Recibe (opcionalmente) el nombre de producto y el nivel de release de la base de datos. Puede utilizar la información recibida en una sentencia <b>case</b>, <b>if</b>, o <b>while</b> de modo que el proceso de ejecución depende de las características de la base de datos.</li></ul>
<code>result = getVAGSysType()</code>	Identifica el sistema destino en el que se ejecuta el programa.

### connectionService()

La función de sistema **VGLib.connectionService** proporciona dos ventajas:

- Permite que un programa se conecte o desconecte a una base de datos durante la ejecución.
- Recibe (opcionalmente) el nombre de producto y el nivel de release de la base de datos. Puede utilizar la información recibida en una sentencia **case**, **if**, o **while** de modo que el proceso de ejecución depende de las características de la base de datos.

Si utiliza **VGLib.connectionService** para crear una conexión nueva desde un programa Java, especifique el nivel de aislamiento estableciendo la variable de sistema **VGVar.sqlIsolationLevel**.

**VGLib.connectionService** sólo puede utilizarse en programas migrados desde VisualAge Generator y EGL 5.0. La función está soportada (durante el desarrollo) si la preferencia EGL **Compatibilidad de VisualAge Generator** está seleccionada o (durante la generación) si la opción del descriptor de construcción **VAGCompatibility** está establecida en *yes*.

Para programas nuevos, utilice las siguientes funciones de sistema:

- SysLib.connect
- SysLib.disconnect
- SysLib.disconnectAll
- SysLib.queryCurrentDatabase
- SysLib.setCurrentDatabase

**VGLib.connectionService** no devuelve ningún valor.

```
VGLib.connectionService(
 idUsuario CHAR(8) in,
 contraseña CHAR(8) in,
 nombreServidor CHAR(18) in
 [, producto CHAR(8) inOut,
 release CHAR(8) inOut
 [, opciónConexión STRING in
])
```

### *idUsuario*

ID de usuario que se utiliza para acceder a la base de datos. El argumento debe ser un elemento de tipo CHAR y longitud 8; un literal no es válido. El argumento es obligatorio. Para obtener información general, consulte la sección *Autorización de base de datos y nombres de tabla*.

### *contraseña*

Contraseña que se utiliza para acceder a la base de datos. El argumento debe ser un elemento de tipo CHAR y longitud 8; un literal no es válido. El argumento es obligatorio.

### *nombreServidor*

Especifica una conexión y utiliza dicha conexión para asignar valores a los argumentos *producto* y *release*, si estos argumentos están incluidos en la invocación de **VGLib.connectionService**.

El argumento *nombreServidor* es obligatorio y debe ser un elemento de tipo CHAR y longitud 18. Son válidos los siguientes valores:

#### **blanks (sin contenido)**

Si una conexión está activa, **VGLib.connectionService** mantiene dicha conexión. Si una conexión no está activa, el resultado (que no sea asignar valores) es volver al estado de conexión que está en vigor al principio de una unidad de ejecución, como se describe en la sección *Base de datos por omisión*.

#### **RESET**

RESET vuelve a conectar a la base de datos por omisión; pero si la base de datos por omisión no está disponible, el estado de conexión sigue igual.

Para obtener más detalles, consulte la sección *Base de datos por omisión*.

### *nombreServidor*

Identifica una base de datos:

- Busque el nombre de la base de datos física consultando la propiedad **vgj.jdbc.database.servidor**, donde *servidor* es el nombre del servidor especificado en la llamada a **VGLib.connectionService**. Si esta propiedad no está definida, el nombre de servidor que está especificado en la llamada a **VGLib.connectionService** se utiliza tal cual.
- El formato del nombre de base de datos es distinto para las conexiones J2EE que para las conexiones no J2EE:
  - Si ha generado el programa para un entorno J2EE, utilice el nombre al que está enlazado el origen de datos en el registro JNDI; por ejemplo, jdbc/MyDB. Esta situación se produce si la opción del descriptor de construcción **J2EE** se ha establecido en YES.
  - Si ha generado el programa para un entorno JDBC no J2EE, utilice un URL de conexión; por ejemplo, jdbc:db2:MyDB. Esta situación se produce si la opción **J2EE** se ha establecido en NO.

### *producto*

Recibe el nombre de producto de la base de datos. El argumento, si existe, debe ser un elemento de tipo CHAR y longitud 8.

Para determinar la serie que se recibirá cuando el código se conecte a una determinada base de datos, revise la documentación del producto para la base de datos o el controlador; o bien ejecute el código en un entorno de prueba y escriba el valor recibido en un archivo.

### *release*

Recibe el nivel de release de la base de datos. El argumento, si existe, debe ser un elemento de tipo CHAR y longitud 8.

Para determinar la serie que se recibirá cuando el código se conecte a una determinada base de datos, revise la documentación del producto para la base de datos o el controlador; o bien ejecute el código en un entorno de prueba y escriba el valor recibido en un archivo.

### *opciónConexión*

Los valores válidos son los siguientes:

#### **D1E**

D1E es el valor por omisión. El *1* en el nombre de la opción indica que sólo se soporta un compromiso de *una* fase y la *E* indica que cualquier desconexión debe tener el valor *explicit*. En este caso, un compromiso o una retrotracción no tiene ningún efecto sobre una conexión existente.

Una conexión a una base de datos no cierra cursores, libera bloqueos ni finaliza una conexión existente. Sin embargo, si la unidad de ejecución ya está conectada a la misma base de datos, el efecto es equivalente a especificar DISC y luego D1E.

Puede utilizar varias conexiones para leer varias bases de datos, pero sólo debe actualizar una base de datos en una unidad de trabajo ya que sólo está disponible un compromiso de una fase.

#### **D1A**

El *1* en el nombre de opción indica que sólo se soporta un compromiso de *una* fase y la *A* indica que cualquier desconexión tiene el valor *automatic*. Las características de esta opción son las siguientes:

- Sólo puede conectarse a una base de datos a la vez
- Un compromiso, retrotracción o conexión a una base de datos finaliza una conexión existente

#### **DISC**

Se desconecta de la base de datos especificada. La desconexión de una base de datos produce una retrotracción y libera los bloqueos, pero sólo para dicha base de datos.

#### **DCURRENT**

Se desconecta de la base de datos conectada actualmente. La desconexión de una base de datos produce una retrotracción y libera los bloqueos, pero sólo para dicha base de datos.

#### **DALL**

Se desconecta de todas las bases de datos conectadas. La desconexión de todas las bases de datos provoca una retrotracción de dichas bases de datos, pero no de otros recursos recuperables.

#### **SET**

Establece una conexión como actual. (Por omisión, la conexión realizada más recientemente en la unidad de ejecución es la actual).

Los siguientes valores están soportados para la compatibilidad con VisualAge Generator, pero son equivalentes a D1E: R, D1C, D2A, D2C, D2E.

**Consideraciones acerca de la definición:** `VGLib.connectionService` establece las siguientes variables de sistema:

- `VGVar.sqlerrd`

- SysVar.sqlca
- SysVar.sqlcode
- VGVar.sqlWarn

#### Ejemplo:

```
VGLib.connectionService(myUserId, myPassword,
 myServerName, myProduct, myRelease, "D1E");
```

#### Conceptos relacionados

- “Diagrama de sintaxis para funciones EGL” en la página 754
- “Unidad lógica de trabajo” en la página 307
- “Unidad de ejecución” en la página 742
- “Soporte de SQL” en la página 229

#### Tareas relacionadas

- “Diagrama de sintaxis para sentencias y mandatos EGL” en la página 755
- “Establecer una conexión JDBC J2EE” en la página 362
- “Cómo se realiza una conexión JDBC estándar” en la página 263

#### Consulta relacionada

- “Autorización de base de datos y nombres de tabla” en la página 466
- “Base de datos por omisión” en la página 251
- “Biblioteca VGLib de EGL” en la página 916

- “Propiedades de ejecución de Java (detalles)” en la página 540
- “sqlDB” en la página 400
- “sqlca” en la página 935
- “sqlcode” en la página 936
- “sqlerrd” en la página 949
- “sqlerrmc” en la página 950
- “sqlIsolationLevel” en la página 950
- “sqlWarn” en la página 951

#### getVAGSysType()

La función de sistema **VGLib.getVAGSysType** identifica el sistema destino en el que se ejecuta el programa. La función está soportada (durante el desarrollo) si la propiedad de programa **VAGCompatibility** está seleccionada o (durante la generación) si la opción del descriptor de construcción **VAGCompatibility** está establecida en *yes*.

Si la salida generada es una envoltura Java, **VGLib.getVAGSysType** no está disponible. De lo contrario, la función devuelve el valor de carácter que devolvería la palabra de función especial EZESYS de VisualAge Generator. Si el sistema actual no está soportado en VisualAge Generator, la función devuelve la serie en mayúsculas equivalente del código devuelto por **SysVar.systemType**.

```
VGLib.getVAGSysType()
returns (resultado CHAR(8))
```

*resultado*

Una serie de caracteres que contiene el código de tipo de sistema, como se muestra en la tabla siguiente.

**VGLib.getVAGSysType** devuelve el equivalente de VisualAge Generator del valor de **SysVar.systemType**.

Valor de sysVar.systemType	Valor devuelto por VGLib.getVAGSysType
AIX	"AIX"
DEBUG	"ITF"
ISERIESC	"OS400"
ISERIESJ	"OS400"
LINUX	"LINUX"
USS	"OS390"
WIN	"WINNT"

El valor devuelto por **VGLib.getVAGSysType** sólo puede utilizarse como serie de caracteres; no puede utilizarse el valor devuelto con los operandos *is* o *not* en una expresión lógica, operación que sí puede realizarse con **sysVar.systemType**:

```
// válido SÓLO para sysVar.systemType
if sysVar.systemType is AIX
 call myProgram;
end
```

El único lugar en el que puede utilizarse **VGLib.getVAGSysType** es como origen de una sentencia assignment o **move**.

Las características de **VGLib.getVAGSysType** son las siguientes:

**Tipo primitivo**

CHAR

**Longitud de datos**

8 (rellenado con blancos)

**¿Se restaura siempre el valor después de una sentencia converse?**

Sí

Es aconsejable utilizar **sysVar.systemType** en lugar de **VGLib.getVAGSysType**.

**Consideraciones acerca de la definición:** El valor de **VGLib.getVAGSysType** no afecta al código que se valida durante la generación. Por ejemplo, la siguiente sentencia **add** se valida aunque se genere para Windows:

```
mySystem CHAR(8);
mySystem = VGLib.getVAGSysType();
if (mySystem == "AIX")
 add myRecord;
end
```

Para evitar la validación de código que nunca se ejecutará en el sistema destino, traslade las sentencias que no desee validar a un segundo programa; a continuación, deje que el programa original llame al programa nuevo de forma condicional:

```
mySystem CHAR(8);
mySystem = VGLib.getVAGSysType();

if (mySystem == "AIX")
 call myAddProgram myRecord;
end
```

Existe una forma alternativa de resolver el problema, pero sólo si utiliza **sysVar.systemType** en lugar de **VGLib.getVAGSysType**; para obtener detalles, consulte *eliminateSystemDependentCode*.

### Consulta relacionada

“Biblioteca VGLib de EGL” en la página 916  
“eliminateSystemDependentCode” en la página 390  
“systemType” en la página 937

---

## Variables de sistema fuera de bibliotecas EGL

Una variable contenida en una biblioteca EGL es global con respecto a la unidad de ejecución. Otras variables de sistema tienen características de ámbito diferentes y se categorizan de la siguiente forma:

### ConverseVar

Variables que resultan de utilidad principalmente en aplicaciones textUI.

### SysVar

Variables que resultan de utilidad para propósitos generales.

### VGVar

Variables que resultan de utilidad principalmente en aplicaciones migradas desde VisualAge Generator.

Si hace referencia a la variable de sistema cuando tiene otro identificador del mismo nombre en el ámbito, debe incluir el nombre de categoría como calificador. Por ejemplo, debe especificar **ConverseVar.eventKey** en lugar de **eventKey** si en el ámbito se encuentra una segunda variable denominada **eventKey**. Si en el ámbito no hay un identificador con el mismo nombre, el calificador es opcional.

### Conceptos relacionados

“Referencias a variables en EGL” en la página 58  
“Reglas de ámbito y “this” en EGL” en la página 56

### Consulta relacionada

“ConverseVar”  
“SysVar” en la página 927  
“VGVar” en la página 940

## ConverseVar

El calificador **ConverseVar** puede preceder al nombre de cada variable de sistema EGL listada en la tabla siguiente. Estas variables resultan de utilidad principalmente en aplicaciones textUI.

Variable de sistema	Descripción
commitOnConverse	Especifica si se produce un compromiso y una liberación de los recursos en una aplicación de texto, antes de que un programa no segmentado emita una operación de inversión (converse). El valor por omisión es 0 (que significa <i>no</i> ) para programas no segmentados y 1 (que significa <i>sí</i> ) para programas segmentados.
eventKey	Identifica la tecla pulsada por el usuario para devolver un formulario a un texto EGL o un .
printerAssociation	Permite especificar, en tiempo de ejecución, el destino de salida cuando imprima un formulario de impresión.

Variable de sistema	Descripción
segmentedMode	Se utiliza en una aplicación de texto para cambiar el efecto de la sentencia converse, pero la variable se pasa por alto con esta finalidad en los programas llamados.
validationMsgNum	Contiene el valor asignado por <b>ConverseLib.validationFailed</b> en una aplicación de texto, a fin de poder determinar si una función de validación ha notificado un error.

### Conceptos relacionados

“Referencias a variables en EGL” en la página 58

“Reglas de ámbito y “this” en EGL” en la página 56

### Consulta relacionada

“Variables de sistema fuera de bibliotecas EGL” en la página 921

## commitOnConverse

La variable de sistema **ConverseVar.commitOnConverse** especifica si se produce un compromiso y una liberación de los recursos en una aplicación de texto, antes de que un programa no segmentado emita una operación de inversión (converse). El valor por omisión es 0 (que significa *no*) para programas no segmentados y 1 (que significa *sí*) para programas segmentados.

Puede utilizar **ConverseVar.commitOnConverse** de las siguientes maneras:

- Como origen o destino de una sentencia assignment o **move**
- Como variable de una expresión lógica utilizada en una sentencia **case**, **if** o **while**
- Como argumento de una sentencia **return** o **exit**.

Otras características de **ConverseVar.commitOnConverse** son las siguientes:

#### Tipo primitivo

NUM

#### Longitud de datos

1

#### ¿Se restaura siempre el valor después de una sentencia converse?

Sí

Para obtener detalles acerca de la utilización de esta variable, consulte el apartado *Segmentación*.

### Conceptos relacionados

“Segmentación en aplicaciones de texto” en la página 160

### Consulta relacionada

“converse” en la página 570

“Variables de sistema fuera de bibliotecas EGL” en la página 921

## eventKey

La variable de sistema **ConverseVar.eventKey** identifica la tecla que el usuario ha pulsado para devolver un formulario a un programa de texto EGL. El valor se restablece cada vez que el programa ejecuta la sentencia **converse**.

Si el código EGL no tiene ningún formulario de entrada, el valor inicial de **ConverseVar.eventKey** es **ENTER**.

Los siguientes valores son válidos (tanto en mayúsculas, minúsculas o una combinación de ambas):

- **ENTER**
- **BYPASS** (que hace referencia a cualquiera de las teclas que se han especificado como teclas de salto para el formulario, o, si no se ha especificado ninguna tecla para el formulario, cualquiera de las teclas que se han especificado como teclas de salto para el formGroup; o si no se ha especificado ninguna tecla para el formGroup, cualquiera de las teclas que se han especificado como teclas de salto para el programa)
- **PA1** a **PA3**
- **PF1** a **PF24** (también se utiliza para F1 a F24)
- **PAKEY** (para cualquier tecla PA)
- **PFKEY** (para cualquier tecla PF o F)

**Nota:** Las teclas **PA** siempre se consideran teclas de salto.

Puede utilizar **ConverseVar.eventKey** como operando de una sentencia **if** o **while**.

Las características de esta variable de sistema son las siguientes:

### Tipo primitivo

CHAR

### Longitud de datos

1

### ¿Se guarda el valor a lo largo de los segmentos?

No

**ConverseVar.eventKey** no es válida en un programa por lotes.

**Ejemplo:** El operador de comparación para **ConverseVar.eventKey** puede ser *is* o *not*, como se muestra en el siguiente ejemplo:

```
if (ConverseVar.eventKey IS PF3)
 exit program(0);
end
```

### Consulta relacionada

“Expresiones lógicas” en la página 497

“Variables de sistema fuera de bibliotecas EGL” en la página 921

## printerAssociation

La variable de sistema **ConverseVar.printerAssociation** permite especificar, durante la ejecución, el destino de la salida al imprimir un formulario de impresión.

Puede utilizar esta variable de cualquiera de estas formas:

- Como origen o destino de una sentencia assignment o move
- Como valor de comparación de una expresión lógica
- Como valor de una sentencia return

Las características de **ConverseVar.printerAssociation** son las siguientes:

**Tipo primitivo**

CHAR

**Longitud de datos**

Varias según el tipo de archivo

**¿Se restaura siempre el valor después de una sentencia converse?**

Sí

**ConverseVar.printerAssociation** se inicializa en el nombre de recurso del sistema especificado durante la generación o para la depuración. Si un programa pasa el control a otro programa, el valor de **ConverseVar.printerAssociation** se establece en el valor por omisión para el programa receptor.

Incluso cuando están permitidos múltiples trabajos de impresión para un formulario de impresión dado, la sentencia close cierra solamente el archivo relacionado con el valor actual de **ConverseVar.printerAssociation**.

**Detalles específicos de la salida de Java:** Para la salida Java, establecerá **ConverseVar.printerAssociation** como una serie de dos partes con dos puntos separadores:

*IDtrabajo:destino*

*IDtrabajo*

Una secuencia de caracteres (sin los dos puntos) que identifica a cada trabajo de impresión de forma exclusiva. Los caracteres son sensibles a las mayúsculas y minúsculas (*job01* es distinto a *JOB01*), y puede volver a utilizar *IDtrabajo* tras cerrarse un trabajo de impresión.

Puede utilizar distintos trabajos para promocionar una clase distinta de salida o un orden de salida distinto, dependiendo del flujo de eventos en el código. Por ejemplo, considere la siguiente secuencia de sentencias EGL:

```
ConverseVar.printerAssociation = "job1";
print form1;
ConverseVar.printerAssociation = "job2";
print form2;
ConverseVar.printerAssociation = "job1";
print form3;
```

Cuando finaliza el programa, se crean dos trabajos de impresión:

- form1 seguido de form3
- form2 solo

*destino* La impresora o archivo que recibe la salida.

La serie *destino* es opcional y se ignora si el trabajo de impresión sigue abierto. Las siguientes sentencias son aplicables si no hay una serie:

- Puede omitir los dos puntos antes de *destino*
- En la mayoría de casos, el programa muestra un diálogo de presentación preliminar desde el que el usuario puede especificar una impresora o un archivo para la salida. La excepción se produce si se utiliza la biblioteca *curses* en UNIX; en ese caso, el trabajo de impresión va a la impresora por omisión.

Las siguientes sentencias corresponden al valor de *destino* al generar para Windows 2000/NT/XP:

- Para enviar salida a la impresora por omisión, haga lo siguiente:

- Especifique un valor que coincida con la propiedad **fileName** en el componente de asociaciones de recursos.
- Cambie las propiedades de ejecución de Java de forma que *spool* (en lugar de *seqws*) sea el valor del tipo de archivo relacionado. Por ejemplo, en el componente de asociaciones de recursos, si el valor de la propiedad **fileName** es *myFile* y el valor de **systemName** es *impresora*, debe cambiar los valores de las propiedades de ejecución de Java de forma que `vgj.ra.myFile.fileType` esté establecido en *spool* en lugar de en *seqws*. Tras el cambio, las propiedades son las siguientes:

```
vgj.ra.myFile.systemName=printer
vgj.ra.myFile.fileType=spool
```

- Para enviar salida a un archivo, especifique un valor que coincida con la propiedad **fileName** en el componente de asociaciones de recursos, cuando *seqws* es el valor de la propiedad **fileType** relacionada en el componente de asociaciones de recursos. La propiedad **systemName** es el componente de asociaciones de recursos que contiene el nombre del archivo del sistema operativo que recibe la salida.
- No especifique el valor *impresora* como el valor de *destino*. Si lo hace, se visualizará el diálogo de presentación preliminar para el usuario, pero ese comportamiento podría cambiar en versiones posteriores de EGL.

Las siguientes sentencias corresponden al valor de *destino* al generar para UNIX:

- Para enviar salida a la impresora por omisión (independientemente de si se está utilizando la biblioteca *curses*), especifique un valor que coincida con la propiedad **fileName** en el componente de asociaciones de recursos, cuando *spool* es el valor de la propiedad **fileType** relacionada en el componente de asociaciones de recursos.
- Para enviar salida a un archivo, especifique un valor que coincida con la propiedad **fileName** en el componente de asociaciones de recursos, cuando *seqws* es el valor de la propiedad **fileType** relacionada en el componente de asociaciones de recursos. La propiedad **systemName** del componente de asociaciones de recursos contiene el nombre del archivo del sistema operativo que recibe la salida.
- No especifique el valor *impresora* como el valor de *destino*. Si lo hace, (y si no se está utilizando la biblioteca *curses*) se visualizará el diálogo de presentación preliminar para el usuario, pero ese comportamiento podría cambiar en versiones posteriores de EGL.

## segmentedMode

La variable de sistema **ConverseVar.segmentedMode** se utiliza en una aplicación de texto para cambiar el efecto de la sentencia *converse*, pero la variable se pasa por alto con esta finalidad en los programas llamados. Para obtener información, consulte el apartado *Segmentación*.

Los valores de **ConverseVar.segmentedMode** son los siguientes:

- 1 La próxima sentencia **converse** se ejecutará en modalidad segmentada.
- 0 La próxima sentencia **converse** se ejecutará en modalidad no segmentada.

El valor por omisión es 0 para programas no segmentados y 1 para programas segmentados. La variable se restablece en el valor por omisión una vez ejecutada la sentencia **converse**.

Puede utilizar esta variable de cualquiera de estas formas:

- Como origen o destino de una sentencia `assignment` o `move`
- Como el valor de cuenta en una sentencia `move ... for count`
- Como valor de comparación de una expresión lógica
- Como valor de una sentencia `return`

Las características de `ConverseVar.segmentedMode` son las siguientes:

**Tipo primitivo**

NUM

**Longitud de datos**

1

**¿Se restaura el valor después de una sentencia `converse`?**

No

**Conceptos relacionados**

“Segmentación en aplicaciones de texto” en la página 160

**Consulta relacionada**

“Variables de sistema fuera de bibliotecas EGL” en la página 921

**validationMsgNum**

La variable del sistema `ConverseVar.validationMsgNum` contiene el valor asignado por `ConverseLib.validationFailed` en una aplicación de texto, a fin de poder determinar si una función de validación ha notificado un error. El valor se restablece a cero en cada uno de los siguientes casos:

- El programa se inicializa
- El programa emite una sentencia de conversión, visualización o impresión
- El programa vuelve a emitir una sentencia de conversión para visualizar un formulario de texto como resultado de un error de validación

Puede utilizar `ConverseVar.validationMsgNum` de las siguientes maneras:

- Como origen o destino de una asignación o sentencia `move` (también permitido en el valor “for count” de una sentencia `move`)
- Como variable de una expresión lógica
- Como argumento de una sentencia `return` o `exit`.

Las características de `ConverseVar.validationMsgNum` son las siguientes:

**Tipo primitivo**

INT

**¿Se restaura siempre el valor después de una sentencia `converse`?**

No

**Ejemplo**

```
/*Conserve el primer número de mensaje definido
durante las rutinas de validación */
if (ConverseVar.validationMsgNum > 0)
 ConverseLib.validationFailed(10);
end
```

**Consulta relacionada**

“converse” en la página 570

“validationFailed()” en la página 791

“display” en la página 573

“print” en la página 632

“Variables de sistema fuera de bibliotecas EGL” en la página 921

## SysVar

El calificador **SysVar** puede preceder al nombre de cada variable de sistema EGL listada en la tabla siguiente. Estas variables resultan de utilidad para propósitos generales.

Variable de sistema	Descripción
arrayIndex	Contiene un número: <ul style="list-style-type: none"><li>• El número del primer elemento de una matriz que coincide con la condición de búsqueda de una expresión lógica simple con un operador <b>in</b>.</li><li>• Cero, si ningún elemento de la matriz coincide con la condición de búsqueda.</li><li>• El número del último elemento modificado en la matriz destino después de una sentencia <b>move ... for count</b>.</li></ul>
callConversionTable	Contiene el nombre de la tabla de conversión utilizada para convertir datos cuando el programa hace lo siguiente durante la ejecución: <ul style="list-style-type: none"><li>• Pasa argumentos durante una llamada a un programa de un sistema remoto</li><li>• Pasa argumentos cuando se invoca un programa remoto mediante la función de sistema <code>sysLib.startTransaction</code></li><li>• Accede a un archivo de una ubicación remota</li></ul>
errorCode	Recibe un código de estado después de alguno de los siguientes eventos: <ul style="list-style-type: none"><li>• La invocación de una sentencia <code>call</code>, si dicha sentencia se encuentra en un bloque <code>try</code></li><li>• Una operación de E/S en un archivo indexado, MQ, relativo o serie</li><li>• La invocación de casi cualquier función de sistema en los siguientes casos:<ul style="list-style-type: none"><li>– La invocación se encuentra dentro de un bloque <code>try</code>; o</li><li>– El programa se ejecuta en modalidad de compatibilidad de VisualAge Generator y <b>VGVar.handleSysLibraryErrors</b> se ha establecido en 1</li></ul></li></ul>

Variable de sistema	Descripción
formConversionTable	<p>Contiene el nombre de la tabla de conversión utilizada para la conversión de texto bidireccional cuando un programa Java generado por EGL actúa del siguiente modo:</p> <ul style="list-style-type: none"> <li>• Muestra un formulario de texto o de impresión que incluye una serie de caracteres hebreos o árabes; o</li> <li>• Muestra un formulario de texto que acepta una serie de caracteres hebreos o árabes del usuario.</li> </ul>
overflowIndicator	Se establece en 1 cuando se produce un desbordamiento aritmético. Al comprobar el valor de esta variable, puede probar las condiciones de desbordamiento.
returnCode	Contiene un código de retorno externo, tal como establece el programa y está disponible en el sistema operativo.
sessionID	Contiene un ID que es específico de la sesión de servidor de aplicaciones Web.
sqlca	Contiene toda el área de comunicaciones de SQL (SQLCA).
sqlcode	Contiene el código de retorno para la operación de E/S SQL finalizada más recientemente. El código se obtiene del área de comunicaciones SQL (SQLCA) y puede variar según el gestor de bases de datos relacionales.
sqlState	Contiene el valor de estado SQL para la operación de E/S SQL finalizada más recientemente. El código se obtiene del área de comunicaciones SQL (SQLCA) y puede variar según el gestor de bases de datos relacionales.
systemType	Identifica el sistema destino en el que se ejecuta el programa.
terminalID	<p>.</p> <p>Se inicializa desde la propiedad del sistema de máquina virtual Java <i>user.name</i> y, si la propiedad no puede recuperarse, está en blanco.</p>
transactionID	Según se describe el tema <i>transactionID</i> .
transferName	Permite especificar, durante la ejecución, el nombre del programa o transacción al que desea realizar la transferencia.
userID	Contiene un identificador de usuario en entornos en los que hay uno disponible.

### Conceptos relacionados

“Referencias a variables en EGL” en la página 58

“Reglas de ámbito y “this” en EGL” en la página 56

## Consulta relacionada

“Variables de sistema fuera de bibliotecas EGL” en la página 921

## arrayIndex

La variable de sistema **SysVar.arrayIndex** contiene un número:

- El número del primer elemento de una matriz que coincide con la condición de búsqueda de una expresión lógica simple con un operador **in**, como se muestra en un ejemplo más adelante.
- Cero, si ningún elemento de la matriz coincide con la condición de búsqueda.
- El número del último elemento modificado en la matriz destino después de una sentencia **move ... for count**.

Puede utilizar **SysVar.arrayIndex** de las siguientes maneras:

- Como subíndice de matriz para acceder a la fila o elemento de matriz coincidente
- Como origen o destino de una sentencia assignment o **move**
- Como el valor de cuenta en una sentencia **move ... for count**
- Como variable de una expresión lógica
- Como argumento de una sentencia **exit** o **return**.

Las características de **SysVar.arrayIndex** son las siguientes:

### Tipo primitivo

BIN

### Longitud de datos

4

### ¿Se restaura siempre el valor después de una sentencia converse?

Sólo en un programa de texto no segmentado; para obtener información detallada, consulte la sección *Segmentación*

**Ejemplo:** Supongamos que el registro *myRecord* se basa en el siguiente componente:

```
Record mySerialRecPart
 serialRecord:
 fileName = "myFile"
 end
 10 zipCodeArray CHAR(9)[100];
 10 cityStateArray CHAR(30)[100];
end
```

Además, supongamos que las matrices se inicializan con códigos postales y combinaciones de ciudad y provincia.

El siguiente código establece la variable *currentCityState* en la ciudad y estado que corresponde al código postal especificado:

```
currentZipCode = "27540";
if (currentZipCode in myRecord.zipCodeArray)
 currentCityState = myRecord.cityStateArray[SysVar.arrayIndex];
end
```

Después de la sentencia **if**, **SysVar.arrayIndex** contiene el índice del primer elemento *zipCodeArray* que contiene el valor de "27540". Si no se encuentra "27540" en *zipCodeArray*, el valor de **SysVar.arrayIndex** es 0.

### Conceptos relacionados

“Segmentación en aplicaciones de texto” en la página 160

### Consulta relacionada

“Matrices” en la página 74

“operador in” en la página 532

“Expresiones lógicas” en la página 497

“Variables de sistema fuera de bibliotecas EGL” en la página 921

### callConversionTable

La variable de sistema **SysVar.callConversionTable** contiene el nombre de la tabla de conversión utilizada para convertir datos cuando el programa hace lo siguiente durante la ejecución:

- Pasa argumentos durante una llamada a un programa de un sistema remoto
- Pasa argumentos cuando se invoca un programa remoto mediante la función de sistema `SysLib.startTransaction`
- Accede a un archivo de una ubicación remota

La conversión se produce cuando los datos se mueven entre sistemas basados en EBCDIC y sistemas basados en ASCII o entre sistemas que utilizan diferentes páginas de códigos. la conversión sólo es posible si el componente de opciones de enlace utilizado durante la generación especifica **PROGRAMCONTROLLED** como valor de la propiedad **conversionTable** de los elementos `callLink` o `asynchLink`. Sin embargo, la conversión no se produce si se especifica **PROGRAMCONTROLLED**, pero no **SysVar.callConversionTable**.

**Características:** Las características de **SysVar.callConversionTable** son las siguientes:

#### Tipo primitivo

CHAR

#### Longitud de datos

8

#### ¿Se guarda el valor a lo largo de los segmentos?

Sí

**Consideraciones de definición:** Debe utilizar **SysVar.callConversionTable** para conmutar entre las tablas de conversión de un programa o para activar o desactivar la conversión de datos en un programa.

**SysVar.callConversionTable** se inicializa en blancos. Para que se produzca la conversión, asegúrese de que el componente de opciones de enlace incluye el valor **PROGRAMCONTROLLED**, como se ha descrito antes, y mueva el nombre de una tabla de conversión a la variable de sistema. Puede establecer

**SysVar.callConversionTable** con un asterisco (\*) para utilizar la tabla de conversión por omisión para el código de idioma nacional por omisión. hace referencia al entorno local por omisión en el sistema destino, siempre que el entorno local esté correlacionado con uno de los idiomas que pueden especificarse para la opción del descriptor de construcción **targetNLS**.

La conversión se realiza en el sistema que origina la llamada, la invocación o el acceso al archivo. Cuando se definen varios niveles de una estructura de registro, la conversión se realiza en los elementos de nivel más bajo (los elementos que no tienen subestructura).

Puede utilizar **SysVar.callConversionTable** de las siguientes maneras:

- Como operando origen o destino de una sentencia assignment o **move**.
- Como variable de una expresión lógica
- Como argumento de una sentencia **return** o **exit**.

El resultado de una comparación de **SysVar.callConversionTable** con otro valor es true sólo si la coincidencia es exacta. Si, por ejemplo, inicializa **SysVar.callConversionTable** con un valor en minúsculas, este valor sólo coincide con un valor en minúsculas.

El valor que se coloca en **SysVar.callConversionTable** permanece sin cambios a efectos de comparación.

#### **Ejemplo:**

```
SysVar.callConversionTable = "ELACNENU";
// tabla de conversión para generación COBOL en inglés de Estados Unidos
```

#### **Consulta relacionada**

“Conversión de datos” en la página 467

“startTransaction()” en la página 911

“Variables de sistema fuera de bibliotecas EGL” en la página 921

“targetNLS” en la página 405

#### **errorCode**

La variable de sistema **SysVar.errorCode** recibe un código de estado después de alguno de los siguientes eventos:

- La invocación de una sentencia call, si dicha sentencia se encuentra en un bloque try
- Una operación de E/S en un archivo indexado, MQ, relativo o serie
- La invocación de casi cualquier función de sistema en los siguientes casos:
  - La invocación se encuentra dentro de un bloque try; o
  - El programa se ejecuta en modalidad de compatibilidad de VisualAge Generator y **VGVar.handleSysLibraryErrors** se ha establecido en 1

Los valores de **SysVar.errorCode** asociados con una función del sistema dada se describen en relación a la función del sistema, no en el tema actual.

Puede utilizar **SysVar.errorCode** de las siguientes maneras:

- Como origen o destino de una sentencia assignment o **move**
- Como variable de una expresión lógica
- En una invocación de función, como argumento asociado con un parámetro in, out o inOut

**SysVar.errorCode** se establece en 0 si la invocación de llamada, E/S o función de sistema es satisfactoria.

Las características de **SysVar.errorCode** son las siguientes:

#### **Tipo primitivo**

CHAR

#### **Longitud de datos**

8

¿Se restaura siempre el valor después de una sentencia converse?

Sí

Para tener una visión general que incluya detalles acerca de **SysVar.errorCode**, consulte la sección *Manejo de excepciones*. La lista de valores posibles de **SysVar.errorCode** se proporciona en *Códigos de error de tiempo de ejecución Java de EGL*.

**Ejemplo:**

```
if (SysVar.errorCode == "00000008")
 exit program;
end
```

**Consulta relacionada**

“Código de error de ejecución de Java EGL” en la página 959

“Manejo de excepciones” en la página 94

“Variables de sistema fuera de bibliotecas EGL” en la página 921

“try” en la página 648

“handleSysLibraryErrors” en la página 947

**formConversionTable**

La variable de sistema **SysVar.formConversionTable** contiene el nombre de la tabla de conversión utilizada para la conversión de texto bidireccional cuando un programa Java generado por EGL actúa del siguiente modo:

- Muestra un formulario de texto o de impresión que incluye una serie de caracteres hebreos o árabes; o
- Muestra un formulario de texto que acepta una serie de caracteres hebreos o árabes del usuario.

**Características:** Las características de **SysVar.formConversionTable** son las siguientes:

**Tipo primitivo**

CHAR

**Longitud de datos**

8

¿Se guarda el valor a lo largo de los segmentos?

Sí

**Consulta relacionada**

“Texto de idioma bidireccional” en la página 470

“Conversión de datos” en la página 467

“Variables de sistema fuera de bibliotecas EGL” en la página 921

**overflowIndicator**

La variable de sistema **SysVar.overflowIndicator** se establece en 1 cuando se produce un desbordamiento aritmético. Al comprobar el valor de esta variable, puede probar las condiciones de desbordamiento.

Después de detectarse una condición de desbordamiento, **SysVar.overflowIndicator** no se restablece automáticamente. Debe incluir código en el programa para restablecer **SysVar.overflowIndicator** en 0 antes de realizar cálculos que puedan desencadenar comprobaciones de desbordamiento.

Puede utilizar **SysVar.overflowIndicator** de las siguientes maneras:

- Como origen o destino de una asignación o sentencia **move** (también permitido en el valor "for count" de una sentencia **move**)
- Como variable de una expresión lógica
- Como argumento de una sentencia **exit** o **return**.

Las características de **SysVar.overflowIndicator** son las siguientes:

**Tipo primitivo**

NUM

**Longitud de datos**

1

**¿Se restaura siempre el valor después de una sentencia converse?**

Sí

**Ejemplo:**

```
SysVar.overflowIndicator = 0;
VGVar.handleOverflow = 2;
a = b;
if (SysVar.overflowIndicator == 1)
 add errorrecord;
end
```

**Consulta relacionada**

"Asignaciones" en la página 374

"Variables de sistema fuera de bibliotecas EGL" en la página 921

"handleOverflow" en la página 946

**returnCode**

La variable de sistema **SysVar.returnCode** contiene un código de retorno externo, tal como establece el programa y está disponible en el sistema operativo. No es posible pasar códigos de retorno de un programa EGL a otro. Por ejemplo, un código de retorno distinto de cero no hace que EGL ejecute un bloque onException.

El valor inicial de **SysVar.returnCode** es cero y el valor debe estar en el rango de -2147483648 a 2147483647, ambos inclusive.

**SysVar.returnCode** sólo es relevante para un programa de texto principal (que se ejecuta fuera de J2EE) o un programa por lotes principal (que se ejecuta fuera de J2EE o en un cliente de aplicaciones J2EE). El objetivo de **SysVar.returnCode** en este contexto es proporcionar un código para el archivo de mandatos o exec que invoca el programa. Si el programa finaliza con un error que no está bajo el control del programa, el entorno de ejecución EGL no tiene en cuenta el valor de **SysVar.returnCode** e intenta devolver el valor 693.

Puede utilizar **SysVar.returnCode** de las siguientes maneras:

- Como origen o destino de una asignación o sentencia **move** (también permitido en el valor "for count" de una sentencia **move**)
- Como variable de una expresión lógica
- Como argumento de una sentencia **exit** o **return**.

Las características de **SysVar.returnCode** son las siguientes:

**Tipo primitivo**

BIN

**Longitud de datos**

9

**¿Se restaura siempre el valor después de una sentencia converse?**

Sí

**Ejemplo:**

```
SysVar.returnValue = 6;
```

**Consulta relacionada**

“Variables de sistema fuera de bibliotecas EGL” en la página 921

**sessionId**

En aplicaciones Web, la variable de sistema **SysVar.sessionID** contiene un ID que es específico de la sesión de servidor de aplicaciones Web. Puede utilizar el valor de **SysVar.sessionID** como valor de clave para acceder a la información de archivo o base de datos que se comparte entre programas.

Fuera de las aplicaciones Web, se aplica lo siguiente:

- La variable de sistema **SysVar.sessionID** contiene un identificador de usuario dependiente del sistema o un identificador de terminal para el programa
- **SysVar.sessionID** sólo soporta uso para la compatibilidad con los productos anteriores a EGL (concretamente, para los releases de CSP anteriores a CSP 370AD Versión 4 Release 1). Es aconsejable utilizar **SysVar.userID** o **SysVar.terminalID** en su lugar.

Puede utilizar **SysVar.sessionID** de las siguientes maneras:

- Como origen de una sentencia assignment o **move**
- Como variable de una expresión lógica
- Como argumento de una sentencia **return**

Las características de **SysVar.sessionID** son las siguientes:

**Tipo primitivo**

CHAR

**Longitud de datos**

8 (se rellena con blancos si el valor tiene menos de 8 caracteres)

**¿Se restaura siempre el valor después de una sentencia converse?**

Sí

**SysVar.sessionID** se inicializa desde la propiedad del sistema de máquina virtual Java *user.name* y, si la propiedad no puede recuperarse, **SysVar.sessionID** está en blanco.

**Ejemplo:**

```
myItem = SysVar.sessionID;
```

**Consulta relacionada**

“Variables de sistema fuera de bibliotecas EGL” en la página 921

“terminalID” en la página 938

“userID” en la página 940

## sqlca

La variable de sistema **SysVar.sqlca** contiene todo el área de comunicaciones SQL (SQLCA). Como se indica más adelante, los valores actuales de un subconjunto de campos de la SQLCA están disponibles para el usuario una vez que el código ha accedido a una base de datos relacional.

Puede utilizar **SysVar.sqlca** de las siguientes maneras:

- Como origen o destino de una sentencia assignment o **move**
- En una invocación de función, como argumento asociado con un parámetro in, out o inOut
- Como variable de una expresión lógica
- Como argumento de una sentencia **exit** o **return**.

Para hacer referencia a campos concretos de SQLCA, debe mover **SysVar.sqlca** a un registro base. El registro debe tener una estructura como se especifica en la descripción de SQLCA para el sistema de gestión de bases de datos. Utilice el registro base si pasa el contenido de SQLCA a un programa remoto de modo que el contenido se convierta correctamente en el formato de datos del sistema remoto.

Para obtener información específica acerca de los campos disponibles en **SysVar.sqlca**, consulte los siguientes temas:

- VGVar.sqlerrd
- SysVar.sqlcode
- SysVar.sqlState
- VGVar.sqlWarn

Las características de **SysVar.sqlca** son las siguientes:

### Tipo primitivo

HEX

### Longitud de datos

272 (136 bytes)

### ¿Se restaura siempre el valor después de una sentencia converse?

Sólo en un programa de texto no segmentado; para obtener información detallada, consulte la sección *Segmentación*

### Ejemplo:

```
myItem = SysVar.sqlca;
```

### Conceptos relacionados

“Segmentación en aplicaciones de texto” en la página 160

“Soporte de SQL” en la página 229

### Consulta relacionada

“Variables de sistema fuera de bibliotecas EGL” en la página 921

“sqlcode” en la página 936

“sqlState” en la página 936

“sqlerrd” en la página 949

“sqlWarn” en la página 951

## sqlcode

La variable de sistema **SysVar.sqlcode** contiene el código de retorno para la operación de E/S SQL finalizada más recientemente. El código se obtiene del área de comunicaciones SQL (SQLCA) y puede variar según el gestor de bases de datos relacionales.

Puede utilizar **SysVar.sqlcode** de las siguientes maneras:

- Como origen o destino de una asignación o sentencia **move** (también permitido en el valor "for count" de una sentencia **move**)
- En una invocación de función, como argumento asociado con un parámetro in, out o inOut
- Como variable de una expresión lógica
- Como argumento de una sentencia **exit** o **return**.

Las características de **SysVar.sqlcode** son las siguientes:

### Tipo primitivo

BIN

### Longitud de datos

9

### ¿Se restaura siempre el valor después de una sentencia converse?

Sólo en un programa de texto no segmentado; para obtener información detallada, consulte la sección *Segmentación*

### Ejemplo:

```
rcitem = SysVar.sqlcode;
```

### Conceptos relacionados

"Segmentación en aplicaciones de texto" en la página 160

"Soporte de SQL" en la página 229

### Consulta relacionada

"Variables de sistema fuera de bibliotecas EGL" en la página 921

## sqlState

La variable de sistema **SysVar.sqlState** contiene el valor de estado SQL de la operación de E/S SQL realizada más recientemente. El código se obtiene del área de comunicaciones SQL (SQLCA) y puede variar según el gestor de bases de datos relacionales.

Puede utilizar **SysVar.sqlState** de las siguientes maneras:

- Como origen o destino de una sentencia assignment o **move**
- En una invocación de función, como argumento asociado con un parámetro in, out o inOut
- Como variable de una expresión lógica
- Como argumento de una sentencia **exit** o **return**

Las características de **SysVar.sqlState** son las siguientes:

### Tipo primitivo

CHAR

## Longitud de datos

5

### ¿Se restaura siempre el valor después de una sentencia `converse`?

Sólo en programas de texto no segmentados; para obtener detalles, consulte el apartado *Segmentación*

### Ejemplo:

```
rcitem = SysVar.sqlState;
```

### Conceptos relacionados

“Segmentación en aplicaciones de texto” en la página 160

“Soporte de SQL” en la página 229

### Consulta relacionada

“Variables de sistema fuera de bibliotecas EGL” en la página 921

## systemType

La variable de sistema **SysVar.systemType** identifica el sistema destino en el que se ejecuta el programa. Si la salida generada es una envoltura Java, **SysVar.systemType** no está disponible. Si no es así, los valores válidos son los siguientes:

**aix** Para AIX

**debug** Para el depurador EGL

**hp** Para HP-UX

**iseriesj**

Para programas iSeries

**linux** Para Linux (en hardware basado en Intel)

**solaris** Para Solaris

**win** Para Windows 2000/NT/XP

Puede utilizar **SysVar.systemType** de las siguientes maneras:

- Como origen de una sentencia `assignment` o **move**
- Como variable de una expresión lógica
- Como argumento de una sentencia **return**

Las características de **SysVar.systemType** son las siguientes:

### Tipo primitivo

CHAR

### Longitud de datos

8 (se rellena con espacios en blanco)

### ¿Se restaura siempre el valor después de una sentencia `converse`?

Sí

Utilice **SysVar.systemType** en lugar de **VGLib.getVAGSysType**.

**Consideraciones de definición:** El valor de **SysVar.systemType** no afecta al código que se valida durante la generación. Por ejemplo, la siguiente sentencia **add** se valida aunque se genere para Windows:

```

if (sysVar.systemType IS AIX)
 add myRecord;
end

```

Para evitar validar código que nunca se ejecutará en el sistema destino, realice una de las siguientes acciones:

- Establezca la opción del descriptor de construcción **EliminateSystemDependentCode** en YES. En este ejemplo, la sentencia **add** no se valida si establece la opción del descriptor de construcción en YES. Sin embargo, tenga cuidado que el generador sólo puede eliminar el código dependiente del sistema si la expresión lógica (en este caso, `SysVar.systemType IS AIX`) es lo suficientemente simple para evaluarse durante la generación.

- Como alternativa, mueva las sentencias que no desea validar a un segundo programa; a continuación, deje que el programa original llame al nuevo programa condicionalmente:

```

if (SysVar.systemType IS AIX)
 call myAddProgram myRecord;
end

```

#### Ejemplo:

```

if (SysVar.systemType is WIN)
 call myAddProgram myRecord;
end

```

#### Consulta relacionada

“eliminateSystemDependentCode” en la página 390

“Variables de sistema fuera de bibliotecas EGL” en la página 921

“getVAGSysType()” en la página 919

## terminalID

.

**SysVar.terminalID** (al igual que **SysVar.sessionID**) se inicializa desde la propiedad del sistema *user.name* de la máquina virtual Java y si la propiedad no puede recuperarse, **SysVar.terminalID** está en blanco.

Puede utilizar **SysVar.terminalID** de las siguientes maneras:

- Como origen de una sentencia assignment o **move**
- Como variable de una expresión lógica
- Como argumento de una sentencia **return**

Las características de **SysVar.terminalID** son las siguientes:

#### Tipo primitivo

CHAR

#### Longitud de datos

8, rellenado con blancos si el valor tiene menos caracteres que el número máximo

¿Se restaura siempre el valor después de una sentencia **converse**?

Sí

#### Ejemplo:

```

myItem10 = SysVar.terminalID;

```

## transactionID

La variable no se utiliza; pero si el programa se ha invocado mediante una sentencia **transfer** con el formato *transferir a programa*, la variable contiene el nombre del programa que realiza la transferencia.

Puede utilizar esta variable de cualquiera de estas formas:

- Como origen o destino de una sentencia assignment o move
- Como valor de comparación de una expresión lógica
- Como valor de una sentencia return

Las características de **SysVar.transactionID** son las siguientes:

### Tipo primitivo

CHAR

### Longitud de datos

8

### ¿Se restaura siempre el valor después de una sentencia converse?

Sí

### Conceptos relacionados

“Segmentación en aplicaciones de texto” en la página 160

### Consulta relacionada

“Variables de sistema fuera de bibliotecas EGL” en la página 921

## transferName

La variable de sistema **SysVar.transferName** permite especificar, durante la ejecución, el nombre del programa o transacción a la que desea realizarse la transferencia.

Puede utilizar esta variable de cualquiera de estas formas:

- Como origen o destino de una sentencia assignment o move
- Como nombre de programa o transacción de una sentencia transfer
- Como valor de comparación de una expresión lógica
- Como valor de una sentencia return

Las características de **SysVar.transferName** son las siguientes:

### Tipo primitivo

CHAR

### Longitud de datos

8

### ¿Se restaura siempre el valor después de una sentencia converse?

Sí

### Consulta relacionada

“Variables de sistema fuera de bibliotecas EGL” en la página 921  
“transfer” en la página 646

## userID

La variable de sistema **SysVar.userID** contiene un identificador de usuario en los entornos donde uno está disponible.

Puede utilizar **SysVar.userID** de las siguientes maneras:

- Como origen de una sentencia assignment o **move**
- Como variable de una expresión lógica
- Como argumento de una sentencia **return**

Las características de **SysVar.userID** son las siguientes:

### Tipo primitivo

CHAR

### Longitud de datos

8 (se rellena con blancos si el valor tiene menos de 8 caracteres)

### ¿Se restaura siempre el valor después de una sentencia converse?

Sí

**SysVar.userID** se inicializa desde la propiedad del sistema de máquina virtual Java *user.name* y, si la propiedad no puede recuperarse, **SysVar.userID** está en blanco.

### Ejemplo:

```
myItem = SysVar.userID;
```

## VGVar

El calificador **VGVar** puede preceder al nombre de cada variable de sistema EGL listada en la tabla siguiente. Estas variables resultan de utilidad principalmente en aplicaciones migradas desde VisualAge Generator.

Variable de sistema	Descripción
currentFormattedGregorianDate	Contiene la fecha del sistema actual en formato gregoriano largo.
currentFormattedJulianDate	Contiene la fecha del sistema actual en formato juliano largo.
currentFormattedTime	Contiene la hora actual del sistema en formato HH:mm:ss.
currentGregorianDate	Contiene la fecha del sistema actual en formato gregoriano de ocho dígitos (aaaaMMdd).
currentJulianDate	Contiene la fecha del sistema actual en formato juliano de siete dígitos (aaaaDDD). Evite utilizar esta variable que existe para soportar la migración de código de VisualAge Generator a EGL.
currentShortGregorianDate	Contiene la fecha del sistema actual en formato gregoriano de seis dígitos (aaMMdd). Evite utilizar esta variable que existe para soportar la migración de código de VisualAge Generator a EGL.

Variable de sistema	Descripción
currentShortJulianDate	Contiene la fecha del sistema actual en formato juliano de cinco dígitos (aaDDD). Evite utilizar esta variable que existe para soportar la migración de código de VisualAge Generator a EGL.
handleHardIOErrors	Controla si un programa continúa ejecutándose después de que se haya producido un error grave en una operación de E/S en un bloque try.
handleOverflow	Controla el proceso de errores después de un desbordamiento aritmético.
handleSysLibraryErrors	Especifica si el valor de la variable de sistema <b>SysVar.errorCode</b> se ve afectado por la invocación de una función de sistema.
mqConditionCode	Contiene el código de finalización de una llamada de API de MQSeries que sigue a una operación de E/S <b>add</b> o <b>scan</b> para un registro MQ.
sqlerrd	Matriz de 6 elementos, en la que cada elemento contiene el valor de área de comunicación SQL (SQLCA) correspondiente devuelto desde la última opción de E/S de SQL.
sqlerrmc	Contiene las variables de sustitución para el mensaje de error asociado al código de retorno de <b>SysVar.sqlcode</b> .
sqlIsolationLevel	Indica el nivel de independencia de una transacción de base de datos con respecto a otra.
sqlWarn	Matriz de 11 elementos, en la que cada elemento contiene un byte de aviso devuelto en el área de comunicaciones SQL (SQLCA) para la última operación de E/S SQL y en la que el índice es superior en uno al número de aviso de la descripción SQLCA SQL.

### Conceptos relacionados

“Referencias a variables en EGL” en la página 58

“Reglas de ámbito y “this” en EGL” en la página 56

### Consulta relacionada

“Variables de sistema fuera de bibliotecas EGL” en la página 921

### currentFormattedGregorianDate

La variable de sistema **VGVar.currentFormattedGregorianDate** contiene la fecha actual del sistema en formato gregoriano largo. El valor se actualiza automáticamente cada vez que el programa hace referencia a la variable de sistema.

El formato está en la siguiente propiedad de entorno de ejecución Java:

```
vgj.datemask.gregorian.long.NLS
```

## NLS

El código NLS (soporte de idioma nacional) especificado en la propiedad de entorno de ejecución Java **vgj.nls.code**. El código es uno de los que se listan en la opción **targetNLS** del descriptor de construcción. Inglés en mayúsculas (código ENP) no está soportado.

Para obtener más detalles acerca de **vgj.nls.code**, consulte el apartado *Propiedades de entorno de ejecución Java (detalles)*.

El formato especificado en **vgj.datemask.gregorian.long.NLS** incluye dd (para día numérico), MM (para mes numérico) y aaaa (para año numérico), utilizándose caracteres distintos de d, M, a o dígitos como separadores. Puede especificar el formato en la opción **dateMask** del descriptor de construcción, y el formato por omisión es específico del entorno local.

Puede utilizar **VGVar.currentFormattedGregorianCalendar** como origen de una sentencia **assignment** o **move** o como argumento de una sentencia **return** o **exit**.

Asegúrese de que este formato de fecha gregoriano largo es el mismo que el formato de fecha especificado para el gestor de base de datos SQL. El emparejamiento de los dos formatos permite a **VGVar.currentFormattedGregorianCalendar** generar fechas en el formato que espera el gestor de base de datos.

Las características de **VGVar.currentFormattedGregorianCalendar** son las siguientes:

### Tipo primitivo

CHAR

### Longitud de datos

10

### ¿Se guarda el valor a lo largo de los segmentos?

No

### Ejemplo:

```
myDate = VGVar.currentFormattedGregorianCalendar;
```

### Conceptos relacionados

“Componente descriptor de construcción” en la página 293

“Propiedades de tiempo de ejecución Java” en la página 347

### Tareas relacionadas

“Editar propiedades de entorno ejecuc. Java en descriptor construcción” en la página 302

### Consulta relacionada

“Biblioteca DateTimeLib de EGL” en la página 791

“Propiedades de ejecución de Java (detalles)” en la página 540

“Variables de sistema fuera de bibliotecas EGL” en la página 921

“targetNLS” en la página 405

## currentFormattedJulianDate

La variable de sistema **VGVar.currentFormattedJulianDate** contiene la fecha actual del sistema en formato juliano largo. El valor se actualiza automáticamente cada vez que el programa hace referencia a la variable de sistema.

El formato está en la siguiente propiedad de entorno de ejecución Java:

```
vgj.datemask.julian.long.NLS
```

## NLS

El código NLS (soporte de idioma nacional) especificado en la propiedad de entorno de ejecución Java **vgj.nls.code**. El código es uno de los que se listan en la opción **targetNLS** del descriptor de construcción. Inglés en mayúsculas (código ENP) no está soportado.

Para obtener más detalles acerca de **vgj.nls.code**, consulte el apartado *Propiedades de entorno de ejecución Java (detalles)*.

El formato especificado en **vgj.datemask.julian.long.NLS** incluye DDD (para día numérico) y aaaa (para año numérico), utilizándose caracteres distintos de D, a o dígitos como separadores. Puede especificar el formato en la opción **dateMask** del descriptor de construcción, y el formato por omisión es específico del entorno local.

Puede utilizar **VGVar.currentFormattedJulianDate** como origen de una sentencia **assignment** o **move** o como argumento de una sentencia **return** o **exit**.

Las características de **VGVar.currentFormattedJulianDate** son las siguientes:

### Tipo primitivo

CHAR

### Longitud de datos

8

### ¿Se guarda el valor a lo largo de los segmentos?

No

Inglés en mayúsculas (código NLS ENP) no está soportado.

### Ejemplo:

```
myDate = VGVar.currentFormattedJulianDate;
```

### Conceptos relacionados

“Componente descriptor de construcción” en la página 293

“Propiedades de tiempo de ejecución Java” en la página 347

### Tareas relacionadas

“Editar propiedades de entorno ejecuc. Java en descriptor construcción” en la página 302

### Consulta relacionada

“Biblioteca DateTimeLib de EGL” en la página 791

“Propiedades de ejecución de Java (detalles)” en la página 540

“Variables de sistema fuera de bibliotecas EGL” en la página 921

“targetNLS” en la página 405

## currentFormattedTime

La variable de sistema **VGVar.currentFormattedTime** contiene la hora actual del sistema en formato HH:mm:ss. El valor se actualiza automáticamente cada vez que el programa hace referencia al mismo.

Puede utilizar **VGVar.currentFormattedTime** de las siguientes maneras:

- Como origen de una sentencia **assignment** o **move**
- Como argumento de una sentencia **exit** o **return**.

Las características de **VGVar.currentFormattedTime** son las siguientes:

**Tipo primitivo**

CHAR

**Longitud de datos**

8

**¿Se guarda el valor a lo largo de los segmentos?**

No

**Ejemplo:**

```
timeField = VGVar.currentFormattedTime;
```

**Consulta relacionada**

"Biblioteca DateTimeLib de EGL" en la página 791

"Variables de sistema fuera de bibliotecas EGL" en la página 921

**currentGregorianDate**

La variable de sistema **VGVar.currentGregorianDate** contiene la fecha actual del sistema en formato gregoriano de ocho dígitos (aaaaMMDD).

El valor de **VGVar.currentGregorianDate** se actualiza automáticamente antes de cada referencia. El valor es numérico y no contiene caracteres separadores.

Puede utilizar **VGVar.currentGregorianDate** como origen de una sentencia **assignment** o **move** o como argumento de una sentencia **return** o **exit**.

Las características de **VGVar.currentGregorianDate** son las siguientes:

**Tipo primitivo**

DATE

**Longitud de datos**

8

**¿Se guarda el valor a lo largo de los segmentos?**

No

**Ejemplo:**

```
myDate = VGVar.currentGregorianDate
```

**Consulta relacionada**

"Biblioteca DateTimeLib de EGL" en la página 791

"Variables de sistema fuera de bibliotecas EGL" en la página 921

**currentJulianDate**

La variable de sistema **VGVar.currentJulianDate** contiene la fecha actual del sistema en formato juliano de siete dígitos (aaaaDDD). Evite utilizar esta variable que existe para soportar la migración de código de VisualAge Generator a EGL.

El valor es numérico, no contiene caracteres de separación y se actualiza automáticamente antes de cada referencia.

Puede utilizar **VGVar.currentJulianDate** como origen de una sentencia **assignment** o **move** o como argumento de una sentencia **return** o **exit**.

Las características de **VGVar.currentJulianDate** son las siguientes:

**Tipo primitivo**

NUM

## Longitud de datos

7

## ¿Se guarda el valor a lo largo de los segmentos?

No

## Ejemplo:

```
myDay = VGVar.currentJulianDate;
```

## Consulta relacionada

“Biblioteca DateTimeLib de EGL” en la página 791

“Variables de sistema fuera de bibliotecas EGL” en la página 921

## currentShortGregorianDate

La variable del sistema **VGVar.currentShortGregorianDate** contiene la fecha del sistema actual en formato gregoriano de seis dígitos (aaMMdd). Evite utilizar esta variable que existe para soportar la migración de código de VisualAge Generator a EGL.

El valor de **VGVar.currentShortGregorianDate** se actualiza automáticamente cada vez que el programa hace referencia al mismo. El valor devuelto es numérico y no contiene caracteres separadores.

Puede utilizar **VGVar.currentShortGregorianDate** como origen de una sentencia **assignment** o **move** o como argumento de una sentencia **return** o **exit**.

Las características de **VGVar.currentShortGregorianDate** son las siguientes:

### Tipo primitivo

NUM

### Longitud de datos

6

## ¿Se guarda el valor a lo largo de los segmentos?

No

## Ejemplo:

```
myDay = VGVar.currentShortGregorianDate;
```

## Consulta relacionada

“Biblioteca DateTimeLib de EGL” en la página 791

“Variables de sistema fuera de bibliotecas EGL” en la página 921

## currentShortJulianDate

La variable de sistema **VGVar.currentShortJulianDate** contiene la fecha actual del sistema en formato juliano de cinco dígitos (aaDDD). Evite utilizar esta variable que existe para soportar la migración de código de VisualAge Generator a EGL.

El valor es numérico, no contiene caracteres de separación y se actualiza automáticamente cada vez que el programa hace referencia al mismo.

Puede utilizar **VGVar.currentShortJulianDate** como origen de una sentencia **assignment** o **move** o como argumento de una sentencia **return** o **exit**.

Las características de **VGVar.currentShortJulianDate** son las siguientes:

**Tipo primitivo**

NUM

**Longitud de datos**

5

**¿Se guarda el valor a lo largo de los segmentos?**

No

**Ejemplo:**

```
myDay = VGVar.currentShortJulianDate;
```

**Consulta relacionada**

"Biblioteca DateTimeLib de EGL" en la página 791

"Variables de sistema fuera de bibliotecas EGL" en la página 921

**handleHardIOErrors**

La variable de sistema **VGVar.handleHardIOErrors** controla si un programa continúa ejecutándose después de que se produzca un error grave en una operación de E/S en un bloque try. El valor por omisión es 1, a menos que establezca la propiedad del programa **handleHardIOErrors** en *no*, lo que establece la variable en 0. (Esta propiedad también está disponible para otros componentes lógicos generables.) Para obtener información, consulte la sección *Manejo de excepciones*.

Puede utilizar **VGVar.handleHardIOErrors** de las siguientes maneras:

- Como origen o destino de una asignación o sentencia **move** (también permitido en el valor "for count" de una sentencia **move**)
- Como variable de una expresión lógica utilizada en una sentencia **case**, **if** o **while**
- Como argumento de una sentencia **return** o **exit**.

Las características de **VGVar.handleHardIOErrors** son las siguientes:

**Tipo primitivo**

NUM

**Longitud de datos**

1

**¿Se restaura siempre el valor después de una sentencia converse?**

Sí

**Ejemplo**

```
VGVar.handleHardIOErrors = 1;
```

**Consulta relacionada**

"Manejo de excepciones" en la página 94

"Variables de sistema fuera de bibliotecas EGL" en la página 921

**handleOverflow**

La variable de sistema **VGVar.handleOverflow** controla el proceso de errores después de un desbordamiento aritmético. Se detectan dos tipos de condiciones de desbordamiento:

- *Desbordamiento de variable de usuario*: se produce cuando el resultado de una operación aritmética o de una asignación a un elemento numérico hace que se pierda un valor significativo (no las posiciones decimales) debido a la longitud del elemento.
- *Desbordamiento de valor máximo*: se produce cuando el resultado de una operación aritmética tiene más de 18 dígitos.

Puede establecer **VGVar.handleOverflow** en uno de los siguientes valores: (El valor por omisión es 0).

Valor	Efecto sobre el desbordamiento de usuario	Efecto sobre el desbordamiento de valor máximo
0	El programa establece la variable de sistema <b>SysVar.overflowIndicator</b> en 1 y continúa	El programa finaliza con un mensaje de error
1	El programa finaliza con un mensaje de error	El programa finaliza con un mensaje de error
2	El programa establece la variable de sistema <b>SysVar.overflowIndicator</b> en 1 y continúa	El programa establece la variable de sistema <b>SysVar.overflowIndicator</b> en 1 y continúa

Puede utilizar **VGVar.handleOverflow** de las siguientes maneras:

- Como origen o destino de una asignación o sentencia **move** (también permitido en el valor "for count" de una sentencia **move**)
- Como variable de una expresión lógica
- Como argumento de una sentencia **exit** o **return**.

Las características de **VGVar.handleOverflow** son las siguientes:

**Tipo primitivo**

NUM

**Longitud de datos**

1

**¿Se restaura siempre el valor después de una sentencia *convert*?**

Sí

**Ejemplo:**

```
VGVar.handleOverflow = 2;
```

**Consulta relacionada**

"Asignaciones" en la página 374

"Variables de sistema fuera de bibliotecas EGL" en la página 921

"overflowIndicator" en la página 932

**handleSysLibraryErrors**

La variable de sistema **VGVar.handleSysLibraryErrors** especifica si el valor de la variable de sistema **SysVar.errorCode** se ve afectado por la invocación de una función de sistema. Sin embargo, **VGVar.handleSysLibraryErrors** sólo está disponible cuando está en vigor la modalidad de compatibilidad de VisualAge Generator, como se describe en el apartado *Compatibilidad con VisualAge Generator*.

Para conocer los detalles y las restricciones, consulte la sección *Manejo de excepciones*.

Puede utilizar **VGVar.handleSysLibraryErrors** de las siguientes maneras:

- Como origen o destino de una sentencia assignment o **move**
- Como variable de una expresión lógica
- Como argumento de una sentencia **exit** o **return**.

Las características de **VGVar.handleSysLibraryErrors** son las siguientes:

**Tipo primitivo**

NUM

**Longitud de datos**

1

**¿Se restaura siempre el valor después de una sentencia converse?**

Sólo en un programa de texto no segmentado; para obtener información detallada, consulte la sección *Segmentación*

**Ejemplo:**

```
VGVar.handleSysLibraryErrors = 1;
```

**Conceptos relacionados**

“Compatibilidad con VisualAge Generator” en la página 439

“Segmentación en aplicaciones de texto” en la página 160

**Consulta relacionada**

“Manejo de excepciones” en la página 94

“Variables de sistema fuera de bibliotecas EGL” en la página 921

“errorCode” en la página 931

**mqConditionCode**

La variable de sistema **VGVar.mqConditionCode** contiene el código de finalización de una llamada de API MQSeries que sigue a una operación de E/S **add** o **scan** para un registro MQ. Los valores válidos y sus significados relacionados son los siguientes:

00 BIEN

01 AVISO

02 ANOMALÍA

Puede utilizar **VGVar.mqConditionCode** de las siguientes maneras:

- Como origen o destino de una asignación o sentencia **move** (también permitido en el valor “for count” de una sentencia **move**)
- Como variable de una expresión lógica
- Como argumento de una sentencia **exit** o **return**.

Las características de **VGVar.mqConditionCode** son las siguientes:

**Tipo primitivo**

NUM

**Longitud de datos**

2

¿Se restaura siempre el valor después de una sentencia `converse`?

Sí

**Ejemplo:**

```
add MQRecord;
if (VGVar.mqConditionCode == 0)
 // continuar
else
 exit program;
end
```

**Conceptos relacionados**

“Soporte de MQSeries” en la página 265

**Consulta relacionada**

“Manejo de excepciones” en la página 94

“Variables de sistema fuera de bibliotecas EGL” en la página 921

**sqlerrd**

La matriz de sistema **VGVar.sqlerrd** es una matriz de 6 elementos, en la que cada elemento contiene el valor de área de comunicación SQL (SQLCA) correspondiente devuelto desde la última operación de E/S SQL. El valor de **VGVar.sqlerrd[3]**, por ejemplo, es el tercer valor e indica el número de filas procesadas para algunas peticiones SQL.

De los elementos de **VGVar.sqlerrd**, sólo **VGVar.sqlerrd[3]** se renueva mediante el sistema de gestión de bases de datos para código Java o durante la depuración.

Puede utilizar un elemento **VGVar.sqlerrd** de las siguientes maneras:

- Como origen o destino de una sentencia `assignment` o **move**
- Como valor de la cláusula `for count` de una sentencia **move**
- En una invocación de función, como argumento asociado con un parámetro `in`, `out` o `inOut`
- Como variable de una expresión lógica
- Como argumento de una sentencia **exit** o **return**.

Las características de cada uno de los elementos de la matriz **VGVar.sqlerrd** son las siguientes:

**Tipo primitivo**

BIN

**Longitud de datos**

9

¿Se restaura siempre el valor después de una sentencia `converse`?

Sólo en un programa de texto no segmentado; para obtener información detallada, consulte la sección *Segmentación*

**Ejemplo:**

```
myItem = VGVar.sqlerrd[3];
```

**Conceptos relacionados**

“Segmentación en aplicaciones de texto” en la página 160

“Soporte de SQL” en la página 229

## Consulta relacionada

“Variables de sistema fuera de bibliotecas EGL” en la página 921

## sqlerrmc

La variable de sistema **VGVar.sqlerrmc** contiene el mensaje de error asociado al código de retorno de **SysVar.sqlcode**. **VGVar.sqlerrmc** se obtiene del área de comunicaciones SQL (SQLCA) y puede variar según el gestor de bases de datos relacionales.

**VGVar.sqlerrmc** no tiene ningún significado para el entorno JDBC.

Puede utilizar **VGVar.sqlerrmc** de las siguientes maneras:

- Como origen o destino de una sentencia assignment o **move**
- Como variable de una expresión lógica
- En una invocación de función, como argumento asociado con un parámetro in, out o inOut
- Como argumento de una sentencia **exit** o **return**.

Las características de **VGVar.sqlerrmc** son las siguientes:

### Tipo primitivo

CHAR

### Longitud de datos

70

### ¿Se restaura siempre el valor después de una sentencia converse?

Sólo en un programa de texto no segmentado; para obtener información detallada, consulte la sección *Segmentación*

### Ejemplo:

```
myItem = VGVar.sqlerrmc;
```

### Conceptos relacionados

“Segmentación en aplicaciones de texto” en la página 160

“Soporte de SQL” en la página 229

## Consulta relacionada

“sqlca” en la página 935

“Variables de sistema fuera de bibliotecas EGL” en la página 921

## sqlIsolationLevel

La variable de sistema **VGVar.sqlIsolationLevel** indica el nivel de independencia de una transacción de base de datos con respecto a otra

Para obtener una visión general del nivel de aislamiento y de las frases *lectura repetible* y *transacción serializable*, consulte la documentación de JDBC disponible en Sun Microsystems, Inc.

**VGVar.sqlIsolationLevel** sólo puede utilizarse en programas migrados desde VisualAge Generator y EGL 5.0. La función está soportada (durante el desarrollo) si la preferencia EGL **Compatibilidad de VisualAge Generator** está seleccionada o (durante la generación) si la opción del descriptor de construcción **VAGCompatibility** está establecida en *yes*.

Para desarrollos nuevos, establezca el nivel de aislamiento de SQL en **SysLib.connect**.

Los siguientes valores de **VGVar.sqlIsolationLevel** están en orden de severidad:

**0 (el valor por omisión)**

Lectura repetible

**1** Transacción serializable

Puede utilizar esta variable de cualquiera de estas maneras:

- Como origen o destino de una sentencia `assignment` o `move`
- En una invocación de función, como argumento asociado con un parámetro `in`, `out` o `inOut`
- Como un valor de comparación en una expresión lógica
- Como el valor en una sentencia de devolución

Las características de **SysVar.transactionID** son las siguientes:

**Tipo primitivo**

NUM

**Longitud de datos**

1

**¿Se restaura el valor siempre después de una conversión?**

Sí

**Consulta relacionada**

“connect()” en la página 895

“Variables de sistema fuera de bibliotecas EGL” en la página 921

**sqlWarn**

La matriz de sistema **VGVar.sqlWarn** es una matriz de 11 elementos, en la que cada elemento contiene un byte de aviso devuelto en el área de comunicaciones SQL (SQLCA) para la última operación de E/S SQL y en la que el índice es superior en uno al número de aviso de la descripción de SQLCA SQL. La variable de sistema **VGVar.sqlWarn[2]**, por ejemplo, hace referencia a SQLWARN1, que indica si los caracteres de un elemento se han truncado en la operación de E/S.

De los elementos de **VGVar.sqlWarn**, sólo la variable de sistema **VGVar.sqlWarn[2]** se renueva mediante el sistema de gestión de bases de datos para código Java o durante la depuración.

Puede utilizar **VGVar.sqlWarn** de las siguientes maneras:

- Como origen o destino de una sentencia `assignment` o `move`
- Como valor de la cláusula `for count` de una sentencia `move`
- Como variable de una expresión lógica
- En una invocación de función, como argumento asociado con un parámetro `in`, `out` o `inOut`
- Como argumento de una sentencia `exit` o `return`.

Las características de cada uno de los elementos de la matriz **VGVar.sqlWarn** son las siguientes:

**Tipo primitivo**

CHAR

## Longitud de datos

1

### ¿Se restaura siempre el valor después de una sentencia converse?

Sólo en un programa de texto no segmentado; para obtener información detallada, consulte la sección *Segmentación*

**Consideraciones de definición:** `VGVar.sqlWarn[2]` contiene *W* si la última operación de E/S SQL ha provocado que el gestor de bases de datos trunque los elementos de datos de carácter debido a espacio insuficiente en las variables de lenguaje principal del programa. Puede utilizar expresiones lógicas para probar si han truncado los valores de determinadas variables del lenguaje principal. Para obtener información detallada, consulte las referencias a **trunc** en la sección *Expresiones lógicas*.

Cuando la variable del lenguaje principal es un número, no se da ningún aviso de truncamiento. Las partes fraccionarias de un número se truncan sin ninguna indicación.

**Ejemplo:** En el siguiente ejemplo, *my-char-field* es un campo del registro de fila SQL que se acaba de procesar y *lost-data* es una función que establece un mensaje de error que indica que se ha truncado la información de *my-char-field*.

```
if (VGVar.sqlWarn[2] == 'W')
 if (my-char-field is trunc)
 lost-data();
 end
end
```

### Conceptos relacionados

“Segmentación en aplicaciones de texto” en la página 160

“Soporte de SQL” en la página 229

### Consulta relacionada

“Expresiones lógicas” en la página 497

“Variables de sistema fuera de bibliotecas EGL” en la página 921

---

## Elemento `transferToTransaction`

Un elemento *transferToTransaction* de un componente de opciones de enlace especifica cómo un programa generado transfiere el control a una transacción y finaliza el proceso. El elemento incluye la propiedad `toPgm` y puede incluir estas propiedades:

- `alias`, necesario si el código se está transfiriendo a un programa cuyo nombre de entorno de ejecución es distinto del nombre del componente de programa relacionado
- `externallyDefined`, necesario si el código se está transfiriendo a un programa no generado con EGL ni VisualAge Generator

Puede evitar la necesidad de especificar un elemento **`transferToTransaction`** si el programa destino se genera con VisualAge Generator o (en ausencia de un alias) con EGL.

### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

#### Tareas relacionadas

“Añadir un componente de opciones de enlace a archivo construcción EGL” en la página 313  
“Editar elementos relacionados con transfer. de comp. opciones enlace” en la página 317

#### Consulta relacionada

“Propiedad alias en elementos de enlace relacionados con la transferencia”  
“Propiedad externallyDefined del elemento transferToTransaction”

## Propiedad alias en elementos de enlace relacionados con la transferencia

En los elementos relacionados con la transferencia del componente de opciones de enlace, la propiedad **alias** especifica el nombre de tiempo de ejecución del programa que se identifica en la propiedad **toPgm**.

El valor de esta propiedad debe coincidir con el alias (si existe) que ha especificado al declarar el programa al que se realiza la transferencia. Si no ha especificado un alias al declarar el programa, establezca la propiedad **alias** en el nombre del componente de programa o no establezca la propiedad en absoluto.

#### Conceptos relacionados

“Componente de opciones de enlace” en la página 311

#### Tareas relacionadas

“Añadir un componente de opciones de enlace a archivo construcción EGL” en la página 313  
“Editar elementos relacionados con transfer. de comp. opciones enlace” en la página 317

#### Consulta relacionada

“Elemento transferToTransaction” en la página 952

## Propiedad externallyDefined del elemento transferToTransaction

El componente de opciones de enlace, elemento transferToTransaction, propiedad **externallyDefined** indica si se está realizando una transferencia a un programa producido por un software que no es EGL o VisualAge Generator. Los valores válidos son **no** (el valor por omisión) y **yes**.

Si especifica **yes**, XCTL implementa la sentencia transfer en todos los sistemas destino COBOL.

Si la propiedad de programa **VAGCompatibility** está establecida en *yes*, puede especificar **externallyDefined** en la sentencia transfer, como se indica en el apartado *Compatibilidad con VisualAge Generator*. Es aconsejable especificar el valor en el elemento transferToTransaction, pero estará en vigor independientemente de dónde se especifique.

#### Conceptos relacionados

“Compatibilidad con VisualAge Generator” en la página 439

#### Tareas relacionadas

“Añadir un componente de opciones de enlace a archivo construcción EGL” en la página 313  
“Editar elementos relacionados con transfer. de comp. opciones enlace” en la página 317

## Declaración use

Esta sección describe la declaración de uso, seguida de los detalles sobre cómo escribir la declaración:

- “En un componente de biblioteca o programa” en la página 955
- “En un componente formGroup” en la página 957
- “En un componente pageHandler” en la página 957

## Información general

La declaración use le permite hacer referencia fácilmente a áreas de datos y funciones en componentes que se generan por separado. Un programa, por ejemplo, puede emitir una declaración use que permita una referencia fácil a una tabla de datos, biblioteca o grupo de formularios, pero solamente si esos componentes son visibles para el componente de programa. Para conocer detalles sobre la visibilidad, consulte *Referencias a componentes*.

En la mayoría de casos, puede hacer referencia a áreas de datos y funciones de otro componente independientemente de si hay una declaración use en vigor. Por ejemplo, si está escribiendo un programa y no tiene una declaración use para un componente de biblioteca denominado `myLib`, puede acceder a la variable de biblioteca denominada `myVar` como se indica a continuación:

```
myLib.myVar
```

Si incluye el nombre de biblioteca en una declaración use, no obstante, puede hacer referencia a la variable como se indica a continuación:

```
myVar
```

La forma abreviada anterior de la referencia solamente es válida si el símbolo `myVar` es exclusivo para cada variable y elemento de estructura que sea global para el programa. (Si el símbolo no es exclusivo, se producirá un error.) Además, el símbolo `myVar` hace referencia a un elemento de la biblioteca solamente si una variable o parámetro locales no tiene el mismo nombre. (Un área de datos local tiene preferencia sobre un área de datos global del programa con el mismo nombre.)

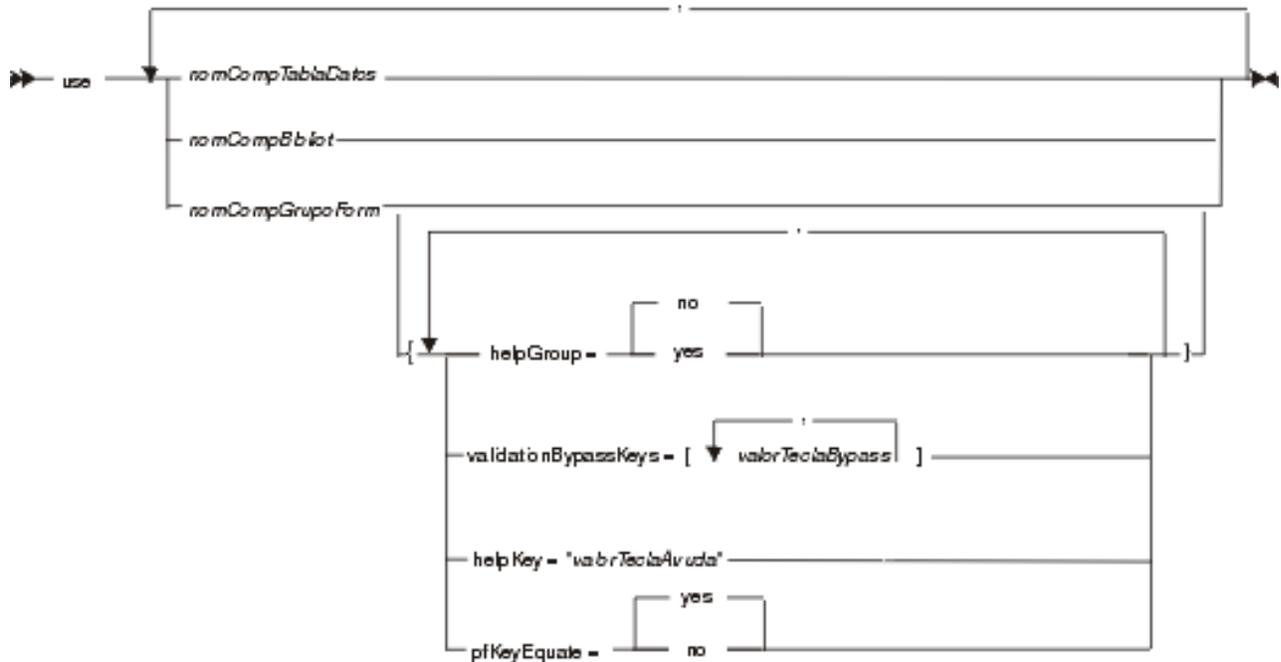
Una declaración use es necesaria en estas situaciones:

- Un programa o biblioteca que utilice alguno de los formularios de un componente `formGroup` dado debe tener una declaración use para ese componente `formGroup`
- Un componente `formGroup` debe tener una declaración use para un formulario que sea necesario para el programa o biblioteca pero que no está incluido en el componente `formGroup`
- Si ha declarado una función en el nivel superior de un archivo fuente EGL en lugar de físicamente dentro de un contenedor (un programa, un `PageHandler` o una biblioteca), esa función puede invocar funciones de biblioteca sólo si se da la situación siguiente:
  - El contenedor incluye una sentencia use que hace referencia a la biblioteca
  - En la función invocante, la propiedad `containerContextDependent` se establece en *yes*

Cada nombre especificado en la declaración use puede estar calificado por un nombre de paquete, un nombre de biblioteca o ambos.

## En un componente de biblioteca o programa

Cada declaración use de un programa o biblioteca debe ser externa a cualquier función. La sintaxis para la declaración es la siguiente:



### *nombreComponenteTablaDatos*

El nombre de un componente dataTable que es visible al programa o biblioteca.

No es necesaria una referencia en una declaración use para un componente dataTable al que se hace referencia en la propiedad del programa **msgTablePrefix**.

No puede alterar temporalmente las propiedades de un componente dataTable en la declaración use.

Para obtener una visión general de los componentes dataTable, consulte *Componente DataTable*.

### *nombreComponenteBiblioteca*

El nombre de un componente de biblioteca que es visible al programa o biblioteca.

No puede alterar temporalmente las propiedades de un componente de biblioteca en la declaración use.

Para obtener una visión general de componentes, consulte las secciones *Componente de biblioteca de tipo basicLibrary* y *Componente de biblioteca de tipo nativeLibrary*.

#### *nombreComponenteGrupoFormularios*

El nombre de un componente formGroup que es visible al programa o biblioteca. Para obtener una visión general de los grupos de formularios, consulte *Componente FormGroup*.

Un programa que utilice alguno de los formularios de un componente formGroup dado debe tener una declaración use para ese componente formGroup.

No se producen alteraciones temporales para las propiedades de nivel de formulario. Si se especifica una propiedad como **validationBypassKeys** en un formulario, por ejemplo, el valor que haya en el formulario estará en vigor durante la ejecución. Si, no obstante, no se especifica una propiedad de nivel de formulario en el formulario, la situación será la siguiente:

- El entorno de ejecución de EGL utiliza el valor de la declaración use del programa
- Si no se ha especificado un valor en la declaración use del programa, el entorno de ejecución de EGL utiliza el valor (si hay alguno) del grupo de formularios

Las propiedades siguientes le permiten cambiar comportamientos cuando un programa específico accede a un grupo de formularios.

#### **helpGroup = no, helpGroup = yes**

Especifica si debe utilizarse el componente formGroup como un grupo de ayuda. El valor por omisión es *no*.

#### **validationBypassKeys = [valorTeclaSalto]**

Identifica una pulsación de usuario que provoca que el entorno de ejecución de EGL se salte las validaciones de campos de entrada. Esta propiedad es de utilidad para reservar una pulsación que finalice el programa rápidamente. Cada opción de *valorTeclaSalto* es como se indica a continuación:

##### **pf*n***

El nombre de una tecla F o PF, incluido un número entre 1 y 24.

**Nota:** Las teclas de función de un teclado de PC suelen ser teclas *f* tales como f1, pero EGL utiliza la terminología de IBM *pf* de forma que, por ejemplo, f1 se denomina pf1.

Si especifica varias claves, separe una de la siguiente mediante una coma.

#### **helpKey = "valorTeclaAyuda"**

Identifica una pulsación de usuario que provoca que el entorno de ejecución de EGL presente al usuario un formulario de ayuda. La opción *valorTeclaAyuda* es como se indica a continuación:

##### **pf*n***

El nombre de una tecla F o PF, incluido un número entre 1 y 24.

**Nota:** Las teclas de función de un teclado de PC suelen ser teclas *f* tales como f1, pero EGL utiliza la terminología de IBM *pf* de forma que, por ejemplo, f1 se denomina pf1.

#### **pfKeyEquate = yes, pfKeyEquate = no**

Especifica si la pulsación que se registra cuando el usuario pulsa una tecla de

función con un número alto (de PF13 a PF24) es la misma que la pulsación registrada cuando el usuario pulsa una tecla de función inferior a 12. El valor por omisión es *yes*. Encontrará los detalles en *pfKeyEquate*.

## En un componente formGroup

En un componente formGroup, una declaración use hace referencia a un formulario especificado fuera del grupo de formularios. Esta clase de declaración permite que múltiples grupos de formularios compartan el mismo formulario.

La sintaxis para una declaración use en un componente formGroup es la siguiente:



*nombreComponenteFormulario*

El nombre de un componente de formulario que es visible al grupo de formularios. Para obtener una visión general de los formularios, consulte *Componente Formulario*.

No puede alterar temporalmente las propiedades de un componente de formulario en la declaración use de un componente formGroup.

## En un componente pageHandler

Cada declaración use de un componente pageHandler debe ser externa a cualquier función. La sintaxis para la declaración es la siguiente:



*nombreComponenteTablaDatos*

El nombre de un componente dataTable que es visible al componente pageHandler.

No puede alterar temporalmente las propiedades de un componente dataTable en la declaración use.

Para obtener una visión general de los componentes dataTable, consulte *Componente DataTable*.

*nombreComponenteBiblioteca*

El nombre de un componente de biblioteca que es visible al componente pageHandler.

No puede alterar temporalmente las propiedades de un componente de biblioteca en la declaración use.

Para obtener una visión general de los componentes de biblioteca, consulte *Componente biblioteca*.

### Conceptos relacionados

“DataTable” en la página 146

“Componente FormGroup” en la página 153

“Componente de formulario” en la página 154

"Componente de biblioteca de tipo basicLibrary" en la página 142  
"Componente de biblioteca de tipo basicLibrary" en la página 142  
"Referencias a componentes" en la página 21

**Consulta relacionada**

"pfKeyEquate" en la página 686

---

## Código de error de ejecución de Java EGL

Cuando se produce un error en el tiempo de ejecución de Java, EGL coloca un código de error en la variable del sistema `sysVar.errorCode` y en la mayoría de los casos presenta un mensaje que tiene el mismo identificador que el código de error. Puede provocar la visualización de un mensaje personalizado en lugar del mensaje EGL; para conocer más detalles, consulte *Personalización de mensajes para el tiempo de ejecución de Java EGL*.

Las situaciones de error son las siguientes:

- Se produce una anomalía durante una llamada remota, una llamada EJB, una operación de comprometer o retrotraer. En esos casos, el identificador de mensaje empieza por CSO.
- Se produce un error en una aplicación Web. En un subconjunto de esos casos, el identificador de mensaje empieza por EGL.
- Se produce un error durante una llamada local, durante el acceso de un archivo o base de datos, o durante la ejecución de una de las siguientes funciones del sistema:
  - Funciones matemáticas
  - Funciones de serie
  - `sysLib.convert`

En esos casos, el identificador de mensaje empieza por VGJ.

- Se produce un error en una función de acceso a Java. En ese caso, el código de error solo incluye números y no se visualiza ningún mensaje.

Los códigos de error asignados por las funciones de acceso a Java se muestran en la tabla siguiente. Los otros códigos de error se muestran en las secciones siguientes.

Valor en <code>sysVar.errorCode</code>	Descripción
00001000	Se ha emitido una excepción por parte de un método invocado o como resultado de una inicialización de clase.
00001001	El objeto era nulo o el identificador especificado no estaba en el espacio de objetos.
00001002	Un método público, campo o clase con el nombre especificado no existe o no puede cargarse.
00001003	El tipo primitivo de EGL no coincide con el tipo esperado en Java.
00001004	El método ha devuelto nulo, el método no devuelve un valor, o el valor de un campo era nulo.
00001005	El valor devuelto no coincide con el tipo del elemento de retorno.
00001006	No se ha podido cargar la clase de un argumento convertido temporalmente a nulo.
00001007	Se ha emitido una <code>SecurityException</code> o <code>IllegalAccessException</code> al intentar obtener información sobre un método o campo, o bien se ha intentado establecer el valor de un campo declarado final.
00001008	No se puede llamar al constructor; el nombre de clase hace referencia a una interfaz o una clase abstracta.

Valor en <code>sysVar.errorCode</code>	Descripción
00001009	Debe especificarse un identificador en lugar de un nombre de clase; el método o campo no es estático.

#### Consulta relacionada

“Valores de error de E/S” en la página 536

“Personalización de mensajes para el tiempo de ejecución de Java EGL” en la página 661  
“errorCode” en la página 931

---

## Código de error de ejecución de Java EGL CSO7000E

**CSO7000E: No se encuentra una entrada para el programa llamado especificado %1 en el archivo de propiedades de enlace %2.**

#### Descripción

El mensaje aparece en esta situación:

- Al generarse el programa de llamada, se estableció la propiedad **remoteBind** en el componente de opciones de enlace, en el elemento **callLink** para el programa llamado; y
- No se encuentra una entrada para el programa llamado especificado en tiempo de ejecución, en el archivo de propiedades de enlace. El motivo puede ser uno de los siguientes:
  - No se encuentra el archivo de propiedades de enlace.
  - Se ha encontrado el archivo, pero una entrada para el programa llamado no está en ese archivo.
  - Se ha especificado un archivo de propiedades de enlace incorrecto.

#### Respuesta del usuario

Haga lo siguiente:

- Si se llama al programa desde una envoltura Java, el archivo de propiedades de enlace debe denominarse *link.properties*, donde *link* es el nombre del componente de opciones de enlace utilizado en la generación. Asegúrese de que el archivo existe, que tiene una entrada para el programa llamado y que está en un directorio o archivador especificado en la variable CLASSPATH.
- Si se llama al programa desde un programa que se ejecuta en el entorno J2EE, el archivo de propiedades de enlace puede identificarse mediante la variable de entorno *cso.linkageOptions.link* en el descriptor de despliegue, donde *link* es el nombre del componente de opciones de enlace utilizado en la generación. Si no se ha establecido la variable de entorno, el archivo de propiedades de enlace debe denominarse *link.properties*, donde *link* es el nombre del componente de opciones de enlace utilizado en la generación. Asegúrese de que el archivo existe, que tiene una entrada para el programa llamado y que está en un directorio o archivador especificado en la CLASSPATH.
- Si se llama al programa desde un programa que no se está ejecutando en el entorno J2EE, la situación es la siguiente:
  - El archivo de propiedades de enlace puede identificarse mediante la propiedad *cso.linkageOptions.link*, donde *link* es el nombre del componente de opciones de enlace utilizado en la generación. Si no se ha establecido la

propiedad, el archivo de propiedades de enlace puede denominarse *link.properties*, donde *link* es el nombre del componente de opciones de enlace utilizado en la generación. En estos dos casos, asegúrese de que el archivo existe, que tiene una entrada para el programa llamado y que está en un directorio o archivador especificado en la CLASSPATH.

- Si no se encuentra el archivo de propiedades de enlace, las propiedades de enlace deben estar en el archivo de propiedades del programa; en ese caso, asegúrese de que el archivo de propiedades del programa incluye una entrada para el programa llamado y que el archivo de propiedades del programa está en un directorio o archivador especificado en CLASSPATH.

Para conocer más detalles, consulte las páginas de ayuda de EGL sobre el elemento **callLink**, sobre las propiedades de ejecución de Java y sobre la configuración del entorno.

Si el problema persiste, haga lo siguiente:

1. Registre el número de mensaje y el texto del mensaje.

**Nota:** El mensaje de error incluye la siguiente información de importancia:

- Dónde se produjo el error
  - El tipo de error interno
2. Registre la situación en la que aparece este mensaje.
  3. Para obtener más instrucciones sobre cómo informar de posibles defectos al Centro de soporte de IBM, consulte la *Guía de instalación de EGL*.

---

## Código de error de ejecución de Java EGL CSO7015E

**CSO7015E: No se puede abrir el archivo de propiedades de enlace %1.**

### Descripción

No se puede abrir el archivo de propiedades de enlace porque el archivo está bloqueado o no se ha encontrado.

### Respuesta del usuario

Asegúrese de que el archivo de propiedades de enlace no está bloqueado por otro proceso y que el archivo reside en un directorio o archivador especificado en su CLASSPATH.

---

## Código de error de ejecución de Java EGL CSO7016E

**CSO7016E: No se ha podido leer el archivo de propiedades csouidpwd.properties. Error: %1**

### Descripción

Se ha encontrado el archivo pero ha habido un error al leerlo.

### Respuesta del usuario

Utilice la parte Error del mensaje para diagnosticar y corregir el problema.

---

## Código de error de ejecución de Java EGL CSO7020E

CSO7020E: La tabla de conversión %1 no es válida.

### Descripción

Una tabla de conversión que maneja texto bidireccional no es válida o no puede cargarse.

### Respuesta del usuario

La tabla de conversión debe residir en un directorio o archivador especificado en la CLASSPATH. Para conocer detalles sobre el desarrollo de la tabla de conversión, consulte la página de ayuda sobre texto bidireccional.

---

## Código de error de ejecución de Java EGL CSO7021E

CSO7021E: El código de atributo de texto de cliente %2 en la tabla de conversión %1 no es válido.

### Descripción

El archivo de tabla de conversión no es válido.

### Respuesta del usuario

Corrija el archivo y vuelva a ejecutar el programa.

---

## Código de error de ejecución de Java EGL CSO7022E

CSO7022E: El código de atributo de texto de servidor %2 en la tabla de conversión %1 no es válido.

### Descripción

El archivo de tabla de conversión no es válido.

### Respuesta del usuario

Corrija el archivo y vuelva a ejecutar el programa.

---

## Código de error de ejecución de Java EGL CSO7023E

CSO7023E: El valor %3 para el código de opción Árabe %2 en la tabla de conversión %1 no es válido.

### Descripción

El archivo de tabla de conversión no es válido.

### Respuesta del usuario

Corrija el archivo y vuelva a ejecutar el programa.

---

## Código de error de ejecución de Java EGL CSO7024E

CSO7024E: El valor %3 para el código de opción Wordbreak %2 en la tabla de conversión %1 no es válido.

### Descripción

El archivo de tabla de conversión no es válido.

### Respuesta del usuario

Corrija el archivo y vuelva a ejecutar el programa.

---

## Código de error de ejecución de Java EGL CSO7026E

CSO7026E: El valor %3 para el código de opción Roundtrip %2 en la tabla de conversión %1 no es válido.

### Descripción

El archivo de tabla de conversión no es válido.

### Respuesta del usuario

Corrija el archivo y vuelva a ejecutar el programa.

---

## Código de error de ejecución de Java EGL CSO7045E

CSO7045E: Error al obtener la dirección de un punto de entrada %1 dentro de la biblioteca compartida %2. RC = %3.

### Descripción

Se ha encontrado un error al obtener la dirección del punto de entrada dentro de la biblioteca compartida.

### Respuesta del usuario

Asegúrese de que la biblioteca compartida referenciada en la biblioteca compartida correcta que debe cargarse. Si es así, asegúrese de que la biblioteca compartida se ha creado correctamente.

---

## Código de error de ejecución de Java EGL CSO7050E

CSO7050E: Se ha producido un error en el programa remoto %1, fecha %2, hora %3

### Descripción

Se ha producido un error en un programa llamado y el programa ha dejado de ejecutarse.

### Respuesta del usuario

Utilice la indicación de la fecha y hora de este mensaje para asociar el mensaje con los mensajes de diagnóstico anotados en la ubicación remota. Compruebe esos

mensajes de diagnóstico para obtener más detalles.

---

## Código de error de ejecución de Java EGL CSO7060E

CSO7060E: Se ha encontrado un error al cargar la biblioteca compartida %1. El código de retorno es %2.

### Descripción

Se ha encontrado un error al cargar la biblioteca compartida.

### Respuesta del usuario

Asegúrese de que la biblioteca compartida reside en un directorio especificado en la variable de entorno PATH o LIBPATH. Asegúrese de que la biblioteca compartida se ha creado correctamente.

---

## Código de error de ejecución de Java EGL CSO7080E

CSO7080E: El protocolo especificado %1 no es válido.

### Descripción

El protocolo especificado en el enlace no se ha reconocido.

### Respuesta del usuario

Consulte la documentación y especifique un protocolo válido.

---

## Código de error de ejecución de Java EGL CSO7160E

CSO7160E: Se ha producido un error en el programa remoto %1, fecha %2, hora %3, en el sistema %4.

### Descripción

El programa Java que está ejecutando llama a un programa remoto en el sistema especificado, que ha fallado en la ejecución en la fecha y hora especificadas.

### Respuesta del usuario

Compruebe las anotaciones del servidor remoto para obtener una descripción más detallada en el análisis de problemas.

---

## Código de error de ejecución de Java EGL CSO7161E

CSO7161E: La unidad de ejecución ha finalizado debido a un error de aplicación en el sistema %1 intentando llamar al programa %2. %3

### Descripción

Se ha producido un error en el servidor remoto que provoca que la unidad de ejecución remota termine de forma anómala al ejecutar el programa remoto. Los mensajes de diagnóstico que preceden a este mensaje en las anotaciones de trabajo del servidor explican la naturaleza del error. Si está disponible, puede haber información adicional incluida con el texto del mensaje.

### Respuesta del usuario

Compruebe los mensajes de error anotados en el sistema servidor para determinar qué hay que hacer para arreglar el problema original.

---

## Código de error de ejecución de Java EGL CSO7162E

**CSO7162E:** Se ha suministrado una contraseña o ID de usuario no válido para la conexión al sistema %1. Recibido el mensaje de excepción de Java: %2.

### Descripción

La contraseña o ID de usuario suministrados para conectarse al sistema remoto no están establecidos o no son válidos.

### Respuesta del usuario

Verifique que está establecida la conexión. Verifique que el ID de usuario y contraseña suministrados al sistema remoto son correctos y vuelva a intentarlo.

---

## Código de error de ejecución de Java EGL CSO7163E

**CSO7163E:** Error de seguridad de acceso remoto al sistema %1 para el usuario %2. Recibido el mensaje de excepción de Java: %3

### Descripción

El usuario especificado que se conecta al sistema actualmente no tiene suficiente autorización o no tiene acceso al recurso remoto en el sistema especificado.

### Respuesta del usuario

Verifique que el usuario que se conecta a la máquina remota tiene la autorización correcta para conectarse a la máquina remota y para ejecutar el programa de servidor remoto.

---

## Código de error de ejecución de Java EGL CSO7164E

**CSO7164E:** Error de conexión remota al sistema %1. Recibido el mensaje de excepción de Java: %2

### Descripción

Se ha producido un error al comunicarse o conectarse al sistema remoto.

### Respuesta del usuario

Compruebe que el servidor remoto está disponible; seguidamente vuelva a intentarlo. Si no resulta, póngase en contacto con el administrador del sistema principal remoto para determinar el problema.

---

## Código de error de ejecución de Java EGL CSO7165E

**CSO7165E:** La operación de comprometer ha fallado en el sistema %1. %2

### Descripción

Ha fallado una operación de comprometer en el sistema remoto.

### Respuesta del usuario

Diagnostique el problema revisando el mensaje detallado, que se muestra aquí como %2.

---

## Código de error de ejecución de Java EGL CSO7166E

CSO7166E: La operación de retrotracción ha fallado en el sistema %1. %2

### Descripción

Ha fallado una operación de retrotracción en el sistema remoto.

### Respuesta del usuario

Diagnostique el problema revisando el mensaje detallado, que se muestra aquí como %2.

---

## Código de error de ejecución de Java EGL CSO7360E

CSO7360E: Error de ejecución de AS400Toolbox: %1, %2 al llamar al programa %3 en el sistema %4

### Descripción

El programa o applet Java que está ejecutando utiliza el protocolo Java400 para llamar a un programa de servidor remoto. Se ha capturado una excepción inesperada al intentar llamar al programa de servidor. El texto del mensaje consta del nombre de la excepción de AS400 Toolbox seguida del mensaje devuelto con la excepción.

### Respuesta del usuario

Utilice el mensaje de error de AS400 Toolbox proporcionado para analizar la causa del problema.

---

## Código de error de ejecución de Java EGL CSO7361E

CSO7361E: Error de Servicios de sistema principal OS/400 EGL. No se han encontrado archivos necesarios en el sistema %1.

### Descripción

El programa o applet Java que está ejecutando utiliza el protocolo Java400 para llamar a un programa de servidor remoto. Se ha activado una excepción al no encontrarse el capturador remoto o no estar en la biblioteca correcta en el servidor.

### Respuesta del usuario

Compruebe que los Servicios de sistema principal OS/400 están instalados correctamente en el sistema remoto. Aplique los PTF más recientes, si están disponibles.

---

## Código de error de ejecución de Java EGL CSO7488E

CSO7488E: Nombre de sistema principal TCP/IP desconocido: %1

### Descripción

Se ha emitido una UnknownHostException durante un intento de conexión al programa de escucha de TCP/IP remoto.

### Respuesta del usuario

Haga lo siguiente:

- Añada la propiedad `cso.serverLinkage.xxx.location` al archivo de propiedades de enlace de ejecución, donde `xxx` es el nombre del programa llamado o es un nombre de aplicación, tal como se describe en la página de ayuda de tipo de referencia EGL en el archivo de propiedades de enlace. El valor de la propiedad es un nombre de sistema principal TCP/IP válido.
- Otra posibilidad es establecer el nombre de sistema principal TCP/IP en el momento de la generación y volver a generar el programa:
  - En el componente de opciones de enlace, en el elemento `callLink` para el programa llamado, establezca la propiedad **location** como el nombre de sistema principal TCP/IP
  - Si desea finalizar las opciones de enlace solamente en la ejecución, establezca la propiedad **remoteBind** como `RUNTIME` y genere con la opción del descriptor de construcción **genProperties** establecida en `YES`

Para conocer más detalles, consulte las páginas de ayuda de EGL sobre el elemento `callLink`, sobre el archivo de propiedades de enlace y sobre la configuración del entorno.

---

## Código de error de ejecución de Java EGL CSO7489E

CSO7489E: La información de enlace utilizada para llamar al programa es incoherente o no está presente.

### Descripción

El programa no ha podido determinar cómo deberá llamarse al programa.

### Respuesta del usuario

Suministre toda la información de enlace necesaria. La información necesaria depende del tipo de llamada deseada. Consulte las páginas de ayuda sobre el componente de opciones de enlace, concretamente sobre el elemento `callLink`.

---

## Código de error de ejecución de Java EGL CSO7610E

CSO7610E: Se ha encontrado un error al llamar a ECI CICS para comprometer una unidad de trabajo. El código de retorno de CICS es %1.

### Descripción

El cliente ha emitido una petición de compromiso pero no ha sido satisfactoria. Se ha encontrado un error al llamar a la interfaz de llamadas externas de CICS para comprometer una unidad de trabajo lógica.

### Respuesta del usuario

Consulte la documentación adecuada de CICS para conocer las acciones correctoras para el error especificado.

---

## Código de error de ejecución de Java EGL CSO7620E

**CSO7620E:** Se ha encontrado un error al llamar al ECI CICS para retrotraer una unidad de trabajo. El código de retorno de CICS es %1.

### Descripción

El cliente ha emitido una petición de retrotracción pero no ha sido satisfactoria. Se ha encontrado un error al llamar a la interfaz de llamadas externas de CICS para retrotraer una unidad de trabajo lógica.

### Respuesta del usuario

Consulte la documentación adecuada de CICS para conocer las acciones correctoras para el error especificado.

---

## Código de error de ejecución de Java EGL CSO7630E

**CSO7630E:** Se ha encontrado un error al finalizar la llamada de procedimiento remoto a un servidor CICS. El código de retorno de CICS es %1.

### Descripción

Se ha intentado comprometer todas las unidades lógicas de trabajo abiertas antes de finalizar la llamada del procedimiento remoto EGL a un servidor CICS pero no ha resultado satisfactorio. Este petición se ha realizado mediante la interfaz de llamadas externas de CICS.

### Respuesta del usuario

Consulte la documentación adecuada de CICS para conocer las acciones correctoras para el error especificado.

---

## Código de error de ejecución de Java EGL CSO7640E

**CSO7640E:** %1 es un valor no válido para la entrada ctgport.

### Descripción

El valor de ctgport debe ser un entero.

### Respuesta del usuario

Utilice el número de ctgport correcto.

---

## Código de error de ejecución de Java EGL CSO7650E

**CSO7650E:** Se ha encontrado un error al llamar al programa %1 utilizando la ECI CICS. Código de retorno: %2. Identificador del sistema CICS: %3.

### Descripción

Se ha devuelto un error de una llamada de función de interfaz de llamadas externas (ECI) de CICS al intentar llamar a un programa de servidor remoto.

El identificador del sistema es el nombre del sistema CICS en el que se debía ejecutar el programa de servidor. Si está en blanco, el sistema se especifica en la definición de programa CICS para el programa o en el archivo de inicialización de cliente CICS. El código de retorno es el código de retorno de CICS.

### Respuesta del usuario

Corrija el problema indicado por el código de retorno.

Para obtener una explicación completa del código de retorno o si el código de retorno no está documentado más arriba, consulte la documentación de ECI de CICS ECI para su sistema para obtener información sobre las acciones correctoras.

Los valores de código de retorno están asociados con símbolos en los archivos de inclusión de ECI CICS faecih.h o cics\_eci.h.

---

## Código de error de ejecución de Java EGL CSO7651E

**CSO7651E: Se ha encontrado un error al llamar al programa %1 utilizando la ECI CICS. Código de retorno: -3 (ECI\_ERR\_NO\_CICS). Identificador del sistema CICS: %2.**

### Descripción

Se ha devuelto un error de una llamada de función de interfaz de llamadas externas (ECI) de CICS al intentar llamar a un programa de servidor remoto.

El identificador del sistema es el nombre del sistema CICS en el que se debía ejecutar el programa de servidor. Si está en blanco, el sistema se especifica en la definición de programa CICS para el programa o en el archivo de inicialización de cliente CICS. El código de retorno es el código de retorno de CICS.

El código de retorno de CICS tiene el siguiente significado:

- -3 - ECI\_ERR\_NO\_CICS  
El sistema servidor o cliente no está disponible

### Respuesta del usuario

Corrija el problema indicado por el código de retorno.

Para obtener una explicación completa del código de retorno o si el código de retorno no está documentado más arriba, consulte la documentación de ECI de CICS ECI para su sistema para obtener información sobre las acciones correctoras.

Los valores de código de retorno están asociados con símbolos en los archivos de inclusión de ECI CICS faecih.h o cics\_eci.h.

---

## Código de error de ejecución de Java EGL CSO7652E

CSO7652E: Se ha encontrado un error al llamar al programa %1 utilizando la ECI CICS. Código de retorno: -4 (ECI\_ERR\_CICS\_DIED). Identificador del sistema CICS: %2.

### Descripción

Se ha devuelto un error de una llamada de función de interfaz de llamadas externas (ECI) de CICS al intentar llamar a un programa de servidor remoto.

El identificador del sistema es el nombre del sistema CICS en el que se debía ejecutar el programa de servidor. Si está en blanco, el sistema se especifica en la definición de programa CICS para el programa o en el archivo de inicialización de cliente CICS. El código de retorno es el código de retorno de CICS.

El código de retorno de CICS tiene el siguiente significado:

- -4 - ECI\_ERR\_CICS\_DIED  
El sistema servidor ya no está disponible

### Respuesta del usuario

Corrija el problema indicado por el código de retorno.

Para obtener una explicación completa del código de retorno o si el código de retorno no está documentado más arriba, consulte la documentación de ECI de CICS ECI para su sistema para obtener información sobre las acciones correctoras.

Los valores de código de retorno están asociados con símbolos en los archivos de inclusión de ECI CICS faecih.h o cics\_eci.h.

---

## Código de error de ejecución de Java EGL CSO7653E

CSO7653E: Se ha encontrado un error al llamar al programa %1 utilizando la ECI CICS. Código de retorno: -6 (ECI\_ERR\_RESPONSE\_TIMEOUT). Identificador del sistema CICS: %2.

### Descripción

Se ha devuelto un error de una llamada de función de interfaz de llamadas externas (ECI) de CICS al intentar llamar a un programa de servidor remoto.

El identificador del sistema es el nombre del sistema CICS en el que se debía ejecutar el programa de servidor. Si está en blanco, el sistema se especifica en la definición de programa CICS para el programa o en el archivo de inicialización de cliente CICS. El código de retorno es el código de retorno de CICS.

El código de retorno de CICS tiene el siguiente significado:

- -6 - ECI\_ERR\_RESPONSE\_TIMEOUT  
Tiempo de respuesta excedido. El límite de tiempo está especificado en la variable de entorno CSOTIMEOUT.

### Respuesta del usuario

Corrija el problema indicado por el código de retorno.

Para obtener una explicación completa del código de retorno o si el código de retorno no está documentado más arriba, consulte la documentación de ECI de CICS ECI para su sistema para obtener información sobre las acciones correctoras.

Los valores de código de retorno están asociados con símbolos en los archivos de inclusión de ECI CICS faacih.h o cics\_eci.h.

---

## Código de error de ejecución de Java EGL CSO7654E

**CSO7654E: Se ha encontrado un error al llamar al programa %1 utilizando la ECI CICS. Código de retorno: -7 (ECI\_ERR\_TRANSACTION\_ABEND). Identificador del sistema CICS: %2. Código Abend: %3.**

### Descripción

Se ha devuelto un error de una llamada de función de interfaz de llamadas externas (ECI) de CICS al intentar llamar a un programa de servidor remoto.

El identificador del sistema es el nombre del sistema CICS en el que se debía ejecutar el programa de servidor. Si está en blanco, el sistema se especifica en la definición de programa CICS para el programa o en el archivo de inicialización de cliente CICS. El código de retorno es el código de retorno de CICS.

El código de retorno de CICS tiene el siguiente significado:

- -7 - ECI\_ERR\_TRANSACTION\_ABEND  
Terminación anómala en el servidor. Los códigos ABEND comunes son:
  - AEI0 - Programa de servidor no definido
  - AEI1 - Transacción de servidor no definida

### Respuesta del usuario

Corrija el problema indicado por el código de retorno.

Para obtener una explicación completa del código de retorno o si el código de retorno no está documentado más arriba, consulte la documentación de ECI de CICS ECI para su sistema para obtener información sobre las acciones correctoras.

Los valores de código de retorno están asociados con símbolos en los archivos de inclusión de ECI CICS faacih.h o cics\_eci.h.

---

## Código de error de ejecución de Java EGL CSO7655E

**CSO7655E: Se ha encontrado un error al llamar al programa %1 utilizando la ECI CICS. Código de retorno: -22 (ECI\_ERR\_UNKNOWN\_SERVER). Identificador del sistema CICS: %2.**

### Descripción

Se ha devuelto un error de una llamada de función de interfaz de llamadas externas (ECI) de CICS al intentar llamar a un programa de servidor remoto.

El identificador del sistema es el nombre del sistema CICS en el que se debía ejecutar el programa de servidor. Si está en blanco, el sistema se especifica en la definición de programa CICS para el programa o en el archivo de inicialización de cliente CICS. El código de retorno es el código de retorno de CICS.

El código de retorno de CICS tiene el siguiente significado:

- -22 - ECI\_ERR\_UNKNOWN\_SERVER  
Sistema de servidor no definido

#### Respuesta del usuario

Corrija el problema indicado por el código de retorno.

Para obtener una explicación completa del código de retorno o si el código de retorno no está documentado más arriba, consulte la documentación de ECI de CICS ECI para su sistema para obtener información sobre las acciones correctoras.

Los valores de código de retorno están asociados con símbolos en los archivos de inclusión de ECI CICS faecih.h o cics\_eci.h.

---

## Código de error de ejecución de Java EGL CSO7656E

**CSO7656E: Se ha encontrado un error al llamar al programa %1 utilizando la ECI CICS. Código de retorno: -27 (ECI\_ERR\_SECURITY\_ERROR). Identificador del sistema CICS: %2.**

#### Descripción

Se ha devuelto un error de una llamada de función de interfaz de llamadas externas (ECI) de CICS al intentar llamar a un programa de servidor remoto.

El identificador del sistema es el nombre del sistema CICS en el que se debía ejecutar el programa de servidor. Si está en blanco, el sistema se especifica en la definición de programa CICS para el programa o en el archivo de inicialización de cliente CICS. El código de retorno es el código de retorno de CICS.

El código de retorno de CICS tiene el siguiente significado:

- -27 - ECI\_ERR\_SECURITY\_ERROR  
ID de usuario o contraseña no válidos

#### Respuesta del usuario

Corrija el problema indicado por el código de retorno.

Para obtener una explicación completa del código de retorno o si el código de retorno no está documentado más arriba, consulte la documentación de ECI de CICS ECI para su sistema para obtener información sobre las acciones correctoras.

Los valores de código de retorno están asociados con símbolos en los archivos de inclusión de ECI CICS faecih.h o cics\_eci.h.

---

## Código de error de ejecución de Java EGL CSO7657E

**CSO7657E: Se ha encontrado un error al llamar al programa %1 utilizando la ECI CICS. Código de retorno: -28 (ECI\_ERR\_MAX\_SYSTEMS). Identificador del sistema CICS: %2.**

### Descripción

Se ha devuelto un error de una llamada de función de interfaz de llamadas externas (ECI) de CICS al intentar llamar a un programa de servidor remoto.

El identificador del sistema es el nombre del sistema CICS en el que se debía ejecutar el programa de servidor. Si está en blanco, el sistema se especifica en la definición de programa CICS para el programa o en el archivo de inicialización de cliente CICS. El código de retorno es el código de retorno de CICS.

El código de retorno de CICS tiene el siguiente significado:

- -28 - ECI\_ERR\_MAX\_SYSTEMS

Se ha alcanzado el número máximo de servidores

### Respuesta del usuario

Corrija el problema indicado por el código de retorno.

Para obtener una explicación completa del código de retorno o si el código de retorno no está documentado más arriba, consulte la documentación de ECI de CICS ECI para su sistema para obtener información sobre las acciones correctoras.

Los valores de código de retorno están asociados con símbolos en los archivos de inclusión de ECI CICS faecih.h o cics\_eci.h.

---

## Código de error de ejecución de Java EGL CSO7658E

**CSO7658E: Se ha encontrado un error al llamar al programa %1 en el sistema %2 para el usuario %3. La llamada de ECI CICS ECI ha devuelto RC %4 y Código Abend %5.**

### Descripción

Se ha devuelto un código de retorno no cero en una llamada de ECI CICS realizada desde la pasarela al sistema especificado en nombre del usuario identificado en el mensaje.

### Respuesta del usuario

Corrija el problema indicado por el código de retorno.

Para obtener una explicación completa del código de retorno, consulte la documentación de ECI de CICS para su sistema para obtener información sobre las acciones correctoras.

Los valores de código de retorno están asociados con símbolos en los archivos de inclusión de ECI CICS faecih.h o cics\_eci.h.

---

## Código de error de ejecución de Java EGL CSO7659E

**CSO7659E: Se ha producido una excepción en el flujo de una petición ECI al sistema CICS %1. Excepción: %2**

### Descripción

Se ha producido una excepción inesperada en el método de flujo al intentar enviar la petición de ECI desde la pasarela al sistema CICS identificado en el mensaje.

### Respuesta del usuario

Examine la serie de excepción que se ha devuelto. Si no puede determinar la causa del problema a partir de la excepción, póngase en contacto con el Soporte de IBM para obtener ayuda.

---

## Código de error de ejecución de Java EGL CSO7669E

**CSO7669E: Se ha encontrado un error al conectarse a CTG. Ubicación de CTG: %1, Puerto de CTG: %2. Excepción: %3**

### Descripción

Se ha producido una excepción inesperada al conectarse a la Pasarela de transacciones CICS.

### Respuesta del usuario

Examine la serie de excepción que se ha devuelto. Si no puede determinar la causa del problema a partir de la excepción, póngase en contacto con el Soporte de IBM para obtener ayuda.

---

## Código de error de ejecución de Java EGL CSO7670E

**CSO7670E: Se ha encontrado un error al desconectarse de CTG. Ubicación de CTG: %1, Puerto de CTG: %2. Excepción: %3**

### Descripción

Se ha producido una excepción inesperada al desconectarse de la Pasarela de transacciones de CICS.

### Respuesta del usuario

Examine la serie de excepción que se ha devuelto. Si no puede determinar la causa del problema a partir de la excepción, póngase en contacto con el Soporte de IBM para obtener ayuda.

---

## Código de error de ejecución de Java EGL CSO7671E

**CSO7671E: Al utilizar el protocolo CICSSSL, deben especificarse ctgKeyStore y ctgKeyStorePassword.**

### Descripción

No se han especificado valores necesarios por lo que no puede completarse la llamada.

### Respuesta del usuario

Asegúrese de que se han especificado ctgKeyStore y ctgKeyStorePassword.

---

## Código de error de ejecución de Java EGL CSO7816E

**CSO7816E: Se ha producido una excepción de socket cuando la pasarela ha intentado conectarse al servidor con el nombre de sistema principal %1 y puerto %2 para el ID de usuario %4. La excepción es: %3**

### Descripción

La llamada de socket para crear y conectar un socket desde la pasarela al sistema servidor identificado en el mensaje ha fallado con la excepción mostrada.

La pasarela EGL ha intentado una llamada de socket para crear y conectar un socket TCP/IP para una llamada de servidor. La llamada de socket ha fallado con la excepción indicada en el mensaje.

### Respuesta del usuario

Examine la información de la excepción para determinar un motivo por el que ha fallado una llamada de socket desde la pasarela. Si no puede determinar la causa del problema examinando la información de la excepción, póngase en contacto con el Soporte de IBM para obtener ayuda.

---

## Código de error de ejecución de Java EGL CSO7819E

**CSO7819E: Se ha producido una excepción inesperada en la función %2. Excepción: %1**

### Descripción

La pasarela EGL ha recibido una excepción inesperada de la función identificada en el mensaje. Puede haberse producido un error interno.

### Respuesta del usuario

Si no puede determinar el origen del problema examinando la información de la excepción, póngase en contacto con el Soporte de IBM para obtener ayuda.

---

## Código de error de ejecución de Java EGL CSO7831E

**CSO7831E: El almacenamiento intermedio del cliente era demasiado pequeño para la cantidad de datos pasados en la llamada. Asegúrese de que el tamaño acumulativo de los parámetros que se pasan no excede el máximo permitido que es de 32567 bytes.**

### Descripción

El almacenamiento intermedio establecido por el cliente no puede ser tan grande como el tamaño acumulativo de los parámetros que se pasan al programa llamado remoto.

### Respuesta del usuario

Asegúrese de que el tamaño acumulativo de los parámetros que se pasan no excede el máximo permitido que es de 32567 bytes. Si no sobrepasan el máximo y se produce este error, informe del mismo al Centro de soporte de IBM.

---

## Código de error de ejecución de Java EGL CSO7836E

CSO7836E: El cliente ha recibido notificación de que el servidor no puede iniciar el programa llamado remoto. Código de razón: %1.

### Descripción

El servidor no puede ejecutar el programa llamado remoto y ha devuelto un código de razón para la determinación de problemas.

### Respuesta del usuario

Los códigos de razón son los siguientes:

- 2 - El servidor no ha podido cargar la clase para el programa llamado. El archivo de rastreo del servidor puede mostrar información más específica. Asegúrese de que la clase está disponible para el servidor.  
Este problema puede dar como resultado una conversión incorrecta del nombre de clase pasado al servidor. Revise la página de ayuda sobre la conversión de datos para verificar que se ha especificado la tabla de conversión correcta en el componente de opciones de enlace, en el elemento callLink para el programa llamado, en la propiedad conversionTable.
- 3 - El programa llamado ha finalizado debido a un error. El archivo de rastreo del servidor puede mostrar información más específica.

Para obtener cualquier código de razón no listado más arriba o si no puede determinar la causa de la anomalía, póngase en contacto con el soporte de IBM.

---

## Código de error de ejecución de Java EGL CSO7840E

CSO7840E: El cliente ha recibido notificación del servidor de que el programa llamado remoto ha fallado con el código de retorno %1.

### Descripción

El programa llamado remoto se ha ejecutado pero ha finalizado con un código de retorno no cero. El problema está en el programa y no en las comunicaciones.

### Respuesta del usuario

Examine o rastree el programa llamado para determinar por qué se ha completado con un código de retorno no cero.

---

## Código de error de ejecución de Java EGL CSO7885E

CSO7885E: Una función de lectura de TCP/IP ha fallado en una llamada para el ID de usuario %2 al nombre de sistema principal %1. La excepción devuelta es: %3

### Descripción

La pasarela EGL ha recibido una excepción al intentar una función de lectura de TCP/IP.

### Respuesta del usuario

Examine la información de la excepción devuelta para poder determinar la causa del problema. Si no puede determinar el motivo de la anomalía, póngase en contacto con el Soporte de IBM para obtener ayuda.

---

## Código de error de ejecución de Java EGL CSO7886E

CSO7886E: Una función de grabación de TCP/IP ha fallado en una llamada para el ID de usuario %2 al nombre de sistema principal %1. La excepción devuelta es: %3

### Descripción

La pasarela EGL ha recibido una excepción al intentar una función de grabación de TCP/IP.

### Respuesta del usuario

Examine la información de la excepción devuelta para poder determinar la causa del problema. Si no puede determinar el motivo de la anomalía, póngase en contacto con el Soporte de IBM para obtener ayuda.

---

## Código de error de ejecución de Java EGL CSO7955E

CSO7955E: %1, %2

### Descripción

Se ha capturado una excepción de Java inesperada.

El texto del mensaje muestra el nombre de la excepción de Java seguido del mensaje de Java que se ha emitido con la excepción.

### Respuesta del usuario

Revise el mensaje y responda como corresponda.

---

## Código de error de ejecución de Java EGL CSO7957E

CSO7957E: El nombre de tabla de conversión %1 no es válido para la conversión de datos de Java.

### Descripción

Está utilizando una clase de Java generada para llamar a un programa y haber especificado incorrectamente una tabla de conversión para convertir datos de Java al formato utilizado por el programa llamado.

### Respuesta del usuario

Revise la página de ayuda sobre la conversión de datos para determinar el nombre de la tabla de conversión, que se especifica en el componente de opciones de enlace, en el elemento callLink para el programa llamado, en la propiedad conversionTable.

---

## Código de error de ejecución de Java EGL CSO7958E

CSO7958E: El código nativo no ha proporcionado un objeto del tipo CSOPowerServer a la envoltura Java, necesario para convertir datos entre la envoltura Java y el programa generado por EGL.

### Descripción

El código Java nativo ha invocado el método call o execute de una envoltura Java sin crear primero una instancia de un objeto de clase CSOPowerServer y proporcionar el objeto a la envoltura.

### Respuesta del usuario

Revise las páginas de ayuda sobre la envoltura Java para obtener detalles sobre el acceso a middleware EGL, que siempre es necesario para la conversión de datos.

---

## Código de error de ejecución de Java EGL CSO7966E

CSO7966E: No se ha encontrado la codificación de página de códigos %1 para la tabla de conversión %2.

### Descripción

La tabla de conversión especificada en las opciones de enlace requiere una codificación no disponible en la máquina virtual Java (JVM) que se utiliza.

### Respuesta del usuario

Revise la página de ayuda sobre la conversión de datos para determinar el nombre de la tabla de conversión correcto, que se especifica en el componente de opciones de enlace, en el elemento callLink para el programa llamado, en la propiedad conversionTable. Si ha especificado la tabla de conversión correcta, asegúrese de que la JVM que está utilizando está soportada por el entorno de ejecución Java de EGL.

Si los pasos anteriores no revelan el problema, considere si la instalación de su JVM no es correcta o si la Máquina virtual Java no da soporte a todas las codificaciones. En estos casos, consulte la documentación de su proveedor de JVM o póngase en contacto con el proveedor de JVM para obtener ayuda.

Si ha encontrado el error al ejecutar un cliente de applets en un navegador, el error se ha producido un el PowerServer SessionManager utilizado por el applet de cliente. En este caso, consulte la documentación de la JVM en la que se ejecuta el SessionManager o póngase en contacto con el proveedor de JVM.

---

## Código de error de ejecución de Java EGL CSO7968E

CSO7968E: El sistema principal %1 no es conocido o no se ha encontrado.

### Descripción

No hay ningún sistema remoto especificado en el enlace.

### Respuesta del usuario

Debe especificarse el sistema remoto en el campo de ubicación del componente de enlace.

---

## Código de error de ejecución de Java EGL CSO7970E

CSO7970E: No se ha podido cargar la biblioteca compartida de EGL %1, razón: %2

### Descripción

La biblioteca compartida es necesaria para completar la operación, pero no se ha podido cargar.

### Respuesta del usuario

Asegúrese de que la biblioteca compartida se encuentra en el sistema. Debe estar incluida en la variable de entorno que especifica la vía de acceso de biblioteca compartida, PATH o LIBPATH.

---

## Código de error de ejecución de Java EGL CSO7975E

CSO7975E: No se ha podido abrir el archivo de propiedades %1.

### Descripción

No se ha podido abrir el archivo de propiedades que necesita el programa. El nombre del archivo de propiedades puede especificarse en la línea de mandatos cuando se inicia el programa. Si no se otorga un nombre cuando se inicia el programa, se utiliza el siguiente nombre por omisión:

```
tcpiplistener.properties
```

El archivo de propiedades no existe, o bien existe pero no se puede abrir.

### Respuesta del usuario

Asegúrese de que el archivo de propiedades existe y que el programa tiene los permisos adecuados para leerlo y, a continuación, vuelva a ejecutar el programa.

---

## Código de error de ejecución de Java EGL CSO7976E

CSO7976E: No se ha podido abrir el archivo de rastreo %1. La excepción es %2. El mensaje es el siguiente: %3

### Descripción

Se ha producido una excepción cuando el programa intentaba abrir el archivo de salida de rastreo.

### Respuesta del usuario

Corrija el problema y vuelva a ejecutar el programa.

---

## Código de error de ejecución de Java EGL CSO7977E

CSO7977E: El archivo de propiedades del programa no contiene un valor válido para la propiedad %1, que es necesaria.

### Descripción

La propiedad no está definida en el archivo de propiedades del programa.

### Respuesta del usuario

Añada la propiedad al archivo de propiedades del programa y vuelva a ejecutar el programa. Encontrará los detalles en la página de ayuda sobre las propiedades del entorno de ejecución Java.

---

## Código de error de ejecución de Java EGL CSO7978E

CSO7978E: Se ha producido una excepción inesperada. La excepción es %1. El mensaje es el siguiente: %2

### Descripción

El programa ha encontrado un error.

### Respuesta del usuario

Corrija el problema y vuelva a ejecutar el programa.

---

## Código de error de ejecución de Java EGL CSO7979E

CSO7979E: No se puede crear un InitialContext. La excepción es %1

### Descripción

La excepción se ha emitido desde el constructor de javax.naming.InitialContext. El programa necesita crear el objeto InitialContext para acceder a los valores del entorno J2EE.

### Respuesta del usuario

Utilice el texto de la excepción y la documentación del entorno J2EE para corregir el problema.

---

## Código de error de ejecución de Java EGL CSO8000E

CSO8000E: La contraseña especificada en la pasarela ha caducado. %1

### Descripción

El GatewayServlet EGL ha recibido una excepción de contraseña caducada al intentar autenticar el usuario con la contraseña suministrada.

### Respuesta del usuario

Examine la información de la excepción devuelta para poder determinar la causa del problema. Corrija el problema proporcionando una nueva contraseña.

---

## Código de error de ejecución de Java EGL CSO8001E

CSO8001E: La contraseña especificada en la pasarela no es válida. %1

### Descripción

El GatewayServlet EGL ha recibido una excepción de contraseña no válida al intentar autenticar el usuario con la contraseña suministrada.

### Respuesta del usuario

Examine la información de la excepción devuelta para poder determinar la causa del problema. Corrija el problema proporcionando una nueva contraseña.

---

## Código de error de ejecución de Java EGL CSO8002E

CSO8002E: El ID de usuario especificado en la pasarela no es válido. %1

### Descripción

El GatewayServlet EGL ha recibido una excepción de ID de usuario no válido al intentar autenticar el usuario con el ID de usuario suministrado.

### Respuesta del usuario

Examine la información de la excepción devuelta para poder determinar la causa del problema. Corrija el problema proporcionando un nuevo ID de usuario.

---

## Código de error de ejecución de Java EGL CSO8003E

CSO8003E: Entrada nula para %1

### Descripción

Se ha detectado una entrada nula.

### Respuesta del usuario

Examine la información de la excepción devuelta para poder determinar la causa del problema. Corrija el problema proporcionando la entrada necesaria.

---

## Código de error de ejecución de Java EGL CSO8004E

CSO8004E: La pasarela ha recibido un error de seguridad desconocido.

### Descripción

El GatewayServlet EGL ha recibido una excepción de seguridad desconocida al intentar autenticar el usuario con la información de usuario suministrada.

### Respuesta del usuario

Examine la información de la excepción devuelta para poder determinar la causa del problema. Corrija el problema proporcionando nueva información de usuario. Si no puede determinar el motivo de la anomalía, póngase en contacto con el Soporte de IBM para obtener ayuda.

---

## Código de error de ejecución de Java EGL CSO8005E

CSO8005E: Se ha producido un error al cambiar la contraseña. %1

### Descripción

El GatewayServlet EGL ha recibido un error al intentar cambiar la contraseña proporcionada.

### Respuesta del usuario

Examine la información de la excepción devuelta para poder determinar la causa del problema. Corrija el problema proporcionando una nueva contraseña. Si no puede determinar el motivo de la anomalía, póngase en contacto con el Soporte de IBM para obtener ayuda.

---

## Código de error de ejecución de Java EGL CSO8100E

CSO8100E: No se puede obtener una fábrica de conexiones. La excepción es %1

### Descripción

La excepción se ha emitido durante una búsqueda de la fábrica de conexiones que se utiliza en una llamada cuando el valor de la propiedad remoteComType es CICSJ2C. La propiedad remoteComType está en el componente de opciones de enlace, en el elemento callLink para el programa llamado.

El nombre de la fábrica de conexiones empieza por java:comp/env/, seguido por el valor que establezca en la propiedad de ubicación del mismo elemento callLink.

### Respuesta del usuario

Asegúrese de que la fábrica de conexiones está definida correctamente en el entorno J2EE y que el valor de la propiedad de ubicación es correcto en el elemento callLink para el programa llamado.

---

## Código de error de ejecución de Java EGL CSO8101E

CSO8101E: No se puede obtener una conexión. La excepción es: %1

### Descripción

La excepción la ha emitido el método getConnection del objeto ConnectionFactory que se ha utilizado para realizar una llamada cuando el valor de la propiedad remoteComType es CICSJ2C. La propiedad remoteComType está en el componente de opciones de enlace, en el elemento callLink para el programa llamado.

### Respuesta del usuario

Es posible que la fábrica de conexiones o el adaptador de recurso no estén definidos o configurados correctamente. Diagnostique el problema consultando el texto de la excepción, la documentación del adaptador de recurso y la documentación del entorno J2EE.

---

## Código de error de ejecución de Java EGL CSO8102E

CSO8102E: No se puede obtener una interacción. La excepción es: %1

### Descripción

La excepción la ha emitido el método createInteraction del objeto Connection que se ha utilizado para realizar una llamada cuando el valor de la propiedad remoteComType es CICSJ2C. La propiedad remoteComType está en el componente de opciones de enlace, en el elemento callLink para el programa llamado.

### Respuesta del usuario

Es posible que la fábrica de conexiones o el adaptador de recurso no estén definidos o configurados correctamente. Diagnostique el problema utilizando el texto de la excepción, la documentación del adaptador de recurso y la documentación del entorno J2EE.

---

## Código de error de ejecución de Java EGL CSO8103E

CSO8103E: No se puede establecer un verbo de interacción. La excepción es %1

### Descripción

La excepción la ha emitido el método setInteractionVerb del objeto ECIInteractionSpec que se ha utilizado para realizar una llamada cuando el valor de la propiedad remoteComType es CICSJ2C. La propiedad remoteComType está en el componente de opciones de enlace, en el elemento callLink para el programa llamado.

### Respuesta del usuario

Es posible que la fábrica de conexiones o el adaptador de recurso no estén definidos o configurados correctamente. Diagnostique el problema utilizando el texto de la excepción, la documentación del adaptador de recurso y la documentación del entorno J2EE.

---

## Código de error de ejecución de Java EGL CSO8104E

CSO8104E: Se ha producido un error durante un intento de comunicar con CICS. La excepción es %1

### Descripción

La excepción la ha emitido el método execute del objeto Interaction que se ha utilizado para realizar una llamada cuando el valor de la propiedad remoteComType es CICSJ2C. La propiedad remoteComType está en el componente de opciones de enlace, en el elemento callLink para el programa llamado.

### Respuesta del usuario

Es posible que la fábrica de conexiones o el adaptador de recurso no estén definidos o configurados correctamente. Diagnostique el problema utilizando el texto de la excepción, la documentación del adaptador de recurso y la documentación del entorno J2EE. Puede haber información adicional en las anotaciones de la pasarela o en un archivo de anotaciones del sistema remoto.

---

## Código de error de ejecución de Java EGL CSO8105E

CSO8105E: No se puede cerrar una interacción o conexión. La excepción es %1

### Descripción

La excepción la ha emitido el método close de un objeto Connection o Interaction que se ha utilizado para realizar una llamada cuando el valor de la propiedad remoteComType es CICSJ2C. La propiedad remoteComType está en el componente de opciones de enlace, en el elemento callLink para el programa llamado.

### Respuesta del usuario

Es posible que la fábrica de conexiones o el adaptador de recurso no estén definidos o configurados correctamente. Diagnostique el problema utilizando el texto de la excepción, la documentación del adaptador de recurso y la documentación del entorno J2EE.

---

## Código de error de ejecución de Java EGL CSO8106E

CSO8106E: No se puede obtener una LocalTransaction para la unidad de trabajo de cliente. La excepción es %1

### Descripción

La excepción la ha emitido el método getLocalTransaction de un objeto Connection que se ha utilizado para realizar una llamada en esta situación:

- El valor de la propiedad remoteComType es CICSJ2C
- El valor de la propiedad luwControl es CLIENT

Esas propiedades están en el componente de opciones de enlace, en el elemento callLink para el programa llamado.

### Respuesta del usuario

Es posible que la fábrica de conexiones o el adaptador de recurso no estén definidos o configurados correctamente. Diagnostique el problema utilizando el texto de la excepción, la documentación del adaptador de recurso y la documentación del entorno J2EE.

---

## Código de error de ejecución de Java EGL CSO8107E

CSO8107E: No se puede establecer el valor de tiempo de espera en una llamada CICSJ2C. La excepción es %1

### Descripción

La excepción la ha emitido el método setExecuteTimeout de un objeto ECIInteractionSpec que se ha utilizado para realizar una llamada cuando el valor de la propiedad remoteComType es CICSJ2C. La propiedad remoteComType está en el componente de opciones de enlace, en el elemento callLink para el programa llamado.

### Respuesta del usuario

Es posible que la fábrica de conexiones o el adaptador de recurso no estén definidos o configurados correctamente. Diagnostique el problema utilizando el texto de la excepción, la documentación del adaptador de recurso y la documentación del entorno J2EE.

---

## Código de error de ejecución de Java EGL CSO8108E

CSO8108E: Se ha producido un error durante un intento de comunicar con CICS.

### Descripción

El método execute del objeto Interaction que se utiliza para la realizar la llamada ha devuelto falso. La llamada no se ha realizado correctamente.

### Respuesta del usuario

Es posible que la fábrica de conexiones o el adaptador de recurso no estén definidos o configurados correctamente. Diagnostique el problema utilizando el texto de la excepción, la documentación del adaptador de recurso y la documentación del entorno J2EE. Puede haber información adicional en las anotaciones de la pasarela o en un archivo de anotaciones del sistema remoto.

---

## Código de error de ejecución de Java EGL CSO8109E

CSO8109E: El valor de tiempo de espera %1 no es válido. Debe ser un número.

### Descripción

Se ha especificado un valor no válido para el tiempo de espera.

### Respuesta del usuario

No especifique un valor de tiempo de espera, o bien especifique un número.

---

## Código de error de ejecución de Java EGL CSO8110E

CSO8110E: La propiedad de enlace parmForm debe establecerse como COMMPTR para llamar al programa %1 ya que hay al menos un parámetro que es una matriz dinámica.

### Descripción

parmForm debe ser COMMPTR ya que uno de los parámetros es una matriz de dinámica.

### Respuesta del usuario

Cambie parmForm a COMMPTR.

---

## Código de error de ejecución de Java EGL CSO8180E

CSO8180E: El enlace ha especificado una llamada DEBUG dentro de un servidor J2EE. La llamada no se ha realizado en un servidor J2EE, el servidor J2EE no está en modalidad de depuración, o bien el servidor J2EE no está habilitado para la depuración de EGL.

### Descripción

La llamada DEBUG no puede completarse.

### Respuesta del usuario

Si la llamada no se realiza en un servidor J2EE, el nombre de sistema principal TCP/IP de la máquina que ejecuta el depurador de EGL debe especificarse en el campo de ubicación del enlace. Si la llamada se realiza en un servidor J2EE, asegúrese de que se inició en modalidad de depuración y asegúrese también de que se han añadido los archivos jar de Depurador de EGL.

---

## Código de error de ejecución de Java EGL CSO8181E

CSO8181E: No se puede contactar con el depurador EGL en el nombre de sistema principal %1 y puerto %2. La excepción es %3

### Descripción

La llamada DEBUG no puede completarse porque no ha podido contactar con el depurador EGL.

### Respuesta del usuario

Asegúrese de que el escucha de EGL está ejecutándose en el depurador de EGL en el nombre de sistema principal y puerto especificados.

---

## Código de error de ejecución de Java EGL CSO8182E

CSO8182E: Se ha producido un error al comunicarse con el depurador de EGL en el nombre de sistema principal %1 y puerto %2. La excepción es %3

### Descripción

La comunicación entre el depurador de EGL y el programa de llamada ha fallado.

### Respuesta del usuario

Utilice la información del mensaje de excepción para corregir el problema.

---

## Código de error de ejecución de Java EGL CSO8200E

CSO8200E: La envoltura de matriz %1 no puede expandirse por encima de su tamaño máximo. El error se ha producido en el método %2.

### Descripción

Se ha excedido el tamaño máximo de la matriz.

### Respuesta del usuario

Compruebe el tamaño y el tamaño máximo de la matriz antes de intentar añadir.

---

## Código de error de ejecución de Java EGL CSO8201E

CSO8201E: %1 es un índice no válido para una envoltura de matriz %2. Tamaño máximo: %3. Tamaño actual: %4

### Descripción

El índice está fuera de los límites de la matriz.

### Respuesta del usuario

Utilice un índice válido.

---

## Código de error de ejecución de Java EGL CSO8202E

CSO8202E: %1 no es un tamaño máximo válido para la envoltura de matriz %2.

### Descripción

La propiedad maxSize debe ser mayor o igual a cero.

### Respuesta del usuario

No establezca la propiedad maxSize en un número negativo.

---

## Código de error de ejecución de Java EGL CSO8203E

CSO8203E: %1 es un tipo de objeto no válido para añadir a una envoltura de matriz de tipo %2.

### Descripción

El contenido de la matriz debe coincidir con su definición.

### Respuesta del usuario

Cambie el tipo de objetos que se almacenan en la matriz, o bien no intente almacenar ese tipo de objeto en la matriz.

---

## Código de error de ejecución de Java EGL CSO8204E

CSO8204E: No se puede pasar una variable Any, Dictionary, ArrayDictionary, Blob, Clob o Ref como un parámetro.

### Descripción

Los tipos listados no pueden utilizarse como parámetros en una sentencia call. Además, los tipos que contienen los tipos listados, tampoco pueden utilizarse como parámetros.

### Respuesta del usuario

No pase esa clase de parámetro al programa llamado.

---

## Código de error de ejecución de Java EGL EGL0650E

EGL0650E: La función %1RequestAttr ha fallado con clave, %2. Error: %3

### Descripción

La función de EGL GetRequestAttr o SetRequestAttr ha fallado al invocarla con la clave dada.

### Respuesta del usuario

Utilice el componente Error de este mensaje para diagnosticar y corregir el problema. Asegúrese de que la función se utiliza dentro de una función de PageHandler.

---

## Código de error de ejecución de Java EGL EGL0651E

EGL0651E: La función %1SessionAttr ha fallado con clave, %2. Error: %3

### Descripción

La función de EGL GetSessionAttr o SetSessionAttr ha fallado al invocarla con la clave dada.

### Respuesta del usuario

Utilice el componente Error de este mensaje para diagnosticar y corregir el problema. Asegúrese de que la función se invoca dentro de una función de PageHandler.

---

## Código de error de ejecución de Java EGL EGL0652E

EGL0652E: La sentencia forward ha fallado con la etiqueta, %1. Error: %2

### Descripción

No se ha podido reenviar el control a la etiqueta dada.

### Respuesta del usuario

Utilice el componente Error de este mensaje para diagnosticar y corregir el problema. Asegúrese de que el objeto EGL asociado con la etiqueta se ha generado correctamente y que la etiqueta está definida en el archivo de configuración de la aplicación.

---

## Código de error de ejecución de Java EGL EGL0653E

EGL0653E: No se ha podido crear un Bean a partir del objeto EGL, %1. Error: %2

### Descripción

No se ha podido crear un bean de acceso a partir del registro EGL o la definición de PageHandler.

### Respuesta del usuario

Utilice el componente Error de este mensaje para diagnosticar y corregir el problema.

---

## Código de error de ejecución de Java EGL EGL0654E

EGL0654E: La función SetError ha fallado con el elemento, %1, clave, %2. Error: %3

### Descripción

La función SetError ha fallado al invocarla con la clave de mensaje dada.

### Respuesta del usuario

Utilice el componente Error de este mensaje para diagnosticar y corregir el problema. Asegúrese de que el elemento tiene una entrada de error en la JSP y que la clave está definida en el archivo de recursos de mensaje.

---

## Código de error de ejecución de Java EGL EGL0655E

EGL0655E: No se han podido copiar datos del Bean al registro EGL, %1. Error: %2

### Descripción

Ha fallado un intento de mover datos desde el bean de formulario al registro.

### Respuesta del usuario

Utilice el componente Error de este mensaje para diagnosticar y corregir el problema. Asegúrese de que la definición del bean coincide con la definición del registro.

---

## Código de error de ejecución de Java EGL EGL0656E

EGL0656E: No se puede asignar la matriz de tamaño %1 a la matriz estática de tamaño %2.

### Descripción

Los tamaños de las matrices deben coincidir.

### Respuesta del usuario

Compruebe las definiciones de matriz de EGL y asegúrese de que los tamaños de matriz son iguales.

---

## Código de error de ejecución de Java EGL EGL0657E

EGL0657E: Ha fallado el proceso de un parámetro de `onPageLoad`. Error: %1.

### Descripción

Se ha producido un error cuando EGL ha intentado recibir valores en los parámetros de la función `onPageLoad`.

### Respuesta del usuario

Utilice el componente Error de este mensaje para diagnosticar y corregir el problema. Asegúrese de que la definición de tipo del valor pasado coincide con el tipo definido para el parámetro en la función `onPageLoad`.

---

## Código de error de ejecución de Java EGL VGJ0001E

VGJ0001E: Desbordamiento de valor máximo de %1.

### Descripción

Durante un cálculo aritmético, se ha dividido un valor entre cero o un resultado intermedio ha excedido 18 dígitos significativos. El programa finaliza a menos que la variable del sistema `VGVar.handleOverflow` esté establecida en 2.

### Respuesta del usuario

Realice una o varias de las siguientes acciones:

- Corrija la lógica del programa para evitar el error.
- Defina la lógica del programa de forma que maneje la condición de desbordamiento; utilice las variables del sistema `VGVar.handleOverflow` y `overflowIndicator`.

---

## Código de error de ejecución de Java EGL VGJ0002E

VGJ0002E: Se ha producido el error %1. No se ha encontrado el texto de mensaje para este error en el archivo de mensajes %2.

### Descripción

El archivo de mensajes podría estar dañado o ser de un release anterior de EGL.

### Respuesta del usuario

Complete una de las siguientes instrucciones:

- Si ha extraído archivos de clase del archivo `fda6.jar`, verifique que las clases que tiene son del mismo release o nivel de mantenimiento que las clases de ese archivo. Si encuentra una discrepancia, sustituya las clases antiguas por la versión correcta.
- Reinstale `fda6.jar` desde EGL.

Si el problema persiste, haga lo siguiente:

1. Registre el número de mensaje y el texto del mensaje.

**Nota:** El mensaje de error incluye la siguiente información de importancia:

- Dónde se produjo el error
  - El tipo de error interno
2. Registre la situación en la que aparece este mensaje.
  3. Para obtener más instrucciones sobre cómo informar de posibles defectos al Centro de soporte de IBM, consulte el manual de instalación del producto.

---

## Código de error de ejecución de Java EGL VGJ0003E

VGJ0003E: Se ha producido un error interno en la ubicación %1.

### Descripción

Este error puede producirse solamente cuando no se han cumplido las restricciones o requisitos del sistema o cuando los componentes de programa EGL se han utilizado incorrectamente. La ubicación especificada en el error se utiliza solamente para fines de diagnóstico de IBM.

### Respuesta del usuario

Compruebe la configuración del programa y reinicie el sistema. Si el problema persiste, haga lo siguiente:

1. Registre el número de mensaje y el texto del mensaje.

**Nota:** El mensaje de error incluye la siguiente información de importancia:

- Dónde se produjo el error
  - El tipo de error interno
2. Registre la situación en la que aparece este mensaje.
  3. Para obtener más instrucciones sobre cómo informar de posibles defectos al Centro de soporte de IBM, consulte el manual de instalación del producto.

---

## Código de error de ejecución de Java EGL VGJ0004I

VGJ0004I: El error se ha producido en %1, función %2.

### Descripción

Este mensaje acompaña a otro mensaje cuando se produce un error. Identifica el programa o registro en el que se ha producido el error, así como la función que se estaba ejecutando en ese momento.

### Respuesta del usuario

Ninguna.

---

## Código de error de ejecución de Java EGL VGJ0005I

VGJ0005I: El error se ha producido en %1.

### Descripción

Este mensaje acompaña a otro mensaje e identifica el programa o registro en el que se ha producido un error.

### Respuesta del usuario

Ninguna.

---

## Código de error de ejecución de Java EGL VGJ0006E

VGJ0006E: Se ha producido un error durante una operación de E/S. %1

### Descripción

Una operación de E/S ha fallado y la sentencia EGL no tiene una sentencia try para tratar el error.

### Respuesta del usuario

Si desea que el programa maneje el error, establezca `handleHardIOErrors` en 1 y coloque la sentencia de E/S en una sentencia try, como en el siguiente ejemplo:

```
VGVar.handleHardIOErrors = 1;

if (userRequest == "A")
 try
 add record1
 onException
 myErrorHandler(12);
 end
end
```

---

## Código de error de ejecución de Java EGL VGJ0007E

VGJ0007E: Desbordamiento del valor mínimo de %1.

### Descripción

La operación aritmética ha generado un resultado que está más allá del valor mínimo permitido para el tipo de datos.

### Respuesta del usuario

Ajuste la expresión aritmética en consecuencia.

---

## Código de error de ejecución de Java EGL VGJ0008E

VGJ0008E: Se ha producido un error de recurso recuperable. %1

### Descripción

Se ha producido un error al cerrar, comprometer o retrotraer un recurso recuperable.

### Respuesta del usuario

Utilice la información del mensaje de error para corregir el problema.

---

## Código de error de ejecución de Java EGL VGJ0009E

VGJ0009E: No se ha podido encontrar ningún campo con el identificador %1 en %2.

### Descripción

Un acceso dinámico ha fallado porque el campo especificado no existe.

### Respuesta del usuario

No acceder a campos no existentes.

---

## Código de error de ejecución de Java EGL VGJ0010E

VGJ0010E: La asignación a %1 ha fallado, el origen de asignación %2 no es compatible.

### Descripción

El tipo del origen no es uno que pueda asignarse al destino.

### Respuesta del usuario

Asegúrese de que los tipos origen y destino sean compatibles al asignar valores.

---

## Código de error de ejecución de Java EGL VGJ0011E

VGJ0011E: No se puede resolver el valor de %1 en un tipo primitivo.

### Descripción

La variable se ha utilizado como un elemento de datos, pero no es un elemento de datos.

### Respuesta del usuario

Cambie el programa para que no utilice la variable como si fuera un elemento de datos.

---

## Código de error de ejecución de Java EGL VGJ0012E

VGJ0012E: No se ha podido evaluar una expresión aritmética; tipos incompatibles en %1.

### Descripción

Los tipos de valores en la expresión son incompatibles.

### Respuesta del usuario

Cambie el programa para utilizar tipos compatibles en la expresión.

---

## Código de error de ejecución de Java EGL VGJ0013E

VGJ0013E: La sentencia de establecimiento ha fallado; no se puede establecer %1 en el estado %2.

### Descripción

El estado especificado no está soportado para la variable.

#### Respuesta del usuario

Cambie el programa para que no intente esta operación.

---

### **Código de error de ejecución de Java EGL VGJ0014E**

VGJ0014E: %1 no puede suscribirse. No es una matriz.

#### Descripción

La variable se ha utilizado como una matriz pero no lo es.

#### Respuesta del usuario

Cambie el programa para que no utilice la variable como una matriz.

---

### **Código de error de ejecución de Java EGL VGJ0015E**

VGJ0015E: %1, %2

#### Descripción

Se ha producido un error. La excepción y el mensaje correspondientes se utilizan como inserciones en este mensaje.

#### Respuesta del usuario

Utilice la información de las inserciones de mensaje para corregir el problema.

---

### **Código de error de ejecución de Java EGL VGJ0016E**

VGJ0016E: No se ha dado un valor a una variable %1.

#### Descripción

La variable se ha utilizando antes de que se le haya asignado un valor.

#### Respuesta del usuario

Cambie el programa para asignar un valor a la variable antes de utilizarla.

---

### **Código de error de ejecución de Java EGL VGJ0017E**

VGJ0017E: La referencia de variable %1 es nula.

#### Descripción

La variable debe hacer referencia a un valor para poder utilizarlo.

#### Respuesta del usuario

Antes de utilizar la variable, déle un valor.

---

## Código de error de ejecución de Java EGL VGJ0018E

VGJ0018E: No se puede realizar un acceso dinámico sobre el registro estructurado %1.

### Descripción

El acceso dinámico no está permitido en un registro estructurado.

### Respuesta del usuario

No utilizar acceso dinámico en el registro estructurado.

---

## Código de error de ejecución de Java EGL VGJ0019E

VGJ0019E: No se puede copiar %1.

### Descripción

Una operación ha intentado copiar algo que no puede copiarse o ha fallado el intento de copia.

### Respuesta del usuario

---

## Código de error de ejecución de Java EGL VGJ0020E

VGJ0020E: La variable llamada %1 no puede utilizarse como %2.

### Descripción

El tipo de variable no permite utilizarla como si fuera del tipo especificado.

### Respuesta del usuario

Cambie el programa para que no se utilice la variable como si fuera un tipo distinto.

---

## Código de error de ejecución de Java EGL VGJ0021E

VGJ0021E: %1 no se puede probar para el estado %2.

### Descripción

El error producido en una expresión IS o NOT. La variable del lado izquierdo de la expresión no soporta el estado especificado como el lado derecho de la expresión.

### Respuesta del usuario

Eliminar o modificar la expresión.

---

## Código de error de ejecución de Java EGL VGJ0050E

VGJ0050E: Se ha producido una excepción al cargar el programa %1. Excepción: %2 Mensaje: %3

### Descripción

No se ha podido cargar la clase del programa.

### Respuesta del usuario

Utilice el mensaje de excepción para diagnosticar y arreglar el problema. La causa más común de este error es que el archivo jar o el directorio que contiene el archivo de clase del programa no esté listado en la variable de entorno CLASSPATH.

---

## Código de error de ejecución de Java EGL VGJ0055E

VGJ0055E: Se ha producido un error en una llamada al programa %1. El código de error es %2 (%3).

### Descripción

El error se ha producido durante una llamada a un programa Java local.

### Respuesta del usuario

Utilice el mensaje de excepción para diagnosticar y arreglar el problema.

---

## Código de error de ejecución de Java EGL VGJ0056E

VGJ0056E: El programa llamado %1 esperaba %2 parámetros pero se han pasado %3.

### Descripción

Se ha pasado un número incorrecto de parámetros a un programa llamado.

### Respuesta del usuario

Vuelva a escribir el programa de llamada o el programa al que se ha llamado, de forma que ambos esperen que se pase el mismo número de parámetros.

---

## Código de error de ejecución de Java EGL VGJ0057E

VGJ0057E: Se ha producido una excepción al pasar parámetros al programa llamado %1. Excepción: %2 Mensaje: %3

### Descripción

Se ha producido un error durante una llamada a un programa Java. El error puede haberse producido antes o después de iniciarse el programa.

### Respuesta del usuario

Utilice la excepción y su mensaje para diagnosticar y arreglar el problema.

---

## Código de error de ejecución de Java EGL VGJ0058E

VGJ0058E: No se ha podido cargar el archivo de propiedades %1.

### Descripción

No se ha podido cargar el archivo de propiedades del programa. El nombre del archivo de propiedades se obtiene de la propiedad del sistema `vgj.properties.file`.

### Respuesta del usuario

Asegúrese de que `vgj.properties.file` tiene el nombre de archivo correcto y de que el archivo de propiedades está en un archivo Jar o un directorio listado en la variable de entorno `CLASSPATH`.

---

## Código de error de ejecución de Java EGL VGJ0060E

**VGJ0060E: StartTransaction a la clase %1 ha fallado. La excepción es %2.**

### Descripción

La excepción se ha emitido mientras el programa estaba intentando iniciar una nueva JVM para ejecutar la clase de servidor especificada como una nueva transacción. La propiedad `vgj.java.command` especifica el mandato utilizado para iniciar una nueva JVM. El mandato por omisión es `java`.

### Respuesta del usuario

Asegúrese de que la propiedad `vgj.java.command` tiene el valor correcto y de que su programa tiene permiso para crear un nuevo proceso.

Coloque la sentencia `startTransaction` dentro de una sentencia `try` para evitar que se este error sea muy grave. Cuando `startTransaction` falle dentro de una sentencia `try`, se guardará un código de error en la variable del sistema `errorCode`.

---

## Código de error de ejecución de Java EGL VGJ0062E

**VGJ0062E: Uno o varios parámetros pasados al programa MQ %1 son de tipo incorrecto. %2**

### Descripción

Se ha emitido una excepción al intentar llamar al programa MQ. Los parámetros son incorrectos.

### Respuesta del usuario

Consulte la documentación del programa MQ y el mensaje de la excepción para corregir el error.

---

## Código de error de ejecución de Java EGL VGJ0064E

**VGJ0064E: El programa %1 esperaba el formulario de texto %2 pero se le entregado el formulario de texto %3 en una sentencia show.**

### Descripción

Ambos programas deben utilizar el mismo formulario de texto.

### Respuesta del usuario

Modifique los programas para utilizar el mismo formulario de texto y vuelva a generar.

---

## Código de error de ejecución de Java EGL VGJ0100E

VGJ0100E: Los datos de %1 no están en formato %2.

### Descripción

Los datos del elemento tienen un formato inesperado. Es posible que se haya grabado otro elemento sobre el elemento especificado.

### Respuesta del usuario

Corrija la lógica del programa para evitar el error.

---

## Código de error de ejecución de Java EGL VGJ0104E

VGJ0104E: %1 no es un índice válido para el subíndice %2 de %3.

### Descripción

Uno de los subíndices utilizados con una matriz multidimensión no es válido. Un valor de subíndice debe estar entre uno y el número de apariciones definidas para el elemento de subíndice.

### Respuesta del usuario

Asegúrese de que el valor de índice es un subíndice válido para el elemento de subíndice.

---

## Código de error de ejecución de Java EGL VGJ0105E

VGJ0105E: %1 no es un índice válido para %2.

### Descripción

Un valor de subíndice debe estar entre uno y el número de apariciones definidas para el elemento de subíndice.

### Respuesta del usuario

Asegúrese de que el valor de índice es un subíndice válido para el elemento de subíndice.

---

## Código de error de ejecución de Java EGL VGJ0106E

VGJ0106E: Desbordamiento de usuario durante la asignación de %1 a %2.

### Descripción

El destino de una asignación no es lo suficientemente grande como para conservar el resultado sin truncar dígitos significativos. El valor de la variable del sistema `VGVar.handleOverflow` es 1, lo que provoca la finalización del programa.

### Respuesta del usuario

Haga lo siguiente:

- Aumente el número de dígitos significativos en el destino; o bien
- Defina la lógica del programa de forma que maneje la condición de desbordamiento; utilice las variables del sistema `VGVar.handleOverflow` y `overflowIndicator`.

---

## Código de error de ejecución de Java EGL VGJ0108E

VGJ0108E: Se ha asignado al elemento HEX %1 el valor no hexadecimal %2.

### Descripción

Los elementos HEX solamente pueden recibir dígitos hexadecimales.

### Respuesta del usuario

Asegúrese de que el valor origen incluye solamente dígitos hexadecimales.

---

## Código de error de ejecución de Java EGL VGJ0109E

VGJ0109E: Se ha asignado al elemento HEX %1 el valor no hexadecimal de %2: %3.

### Descripción

Los elementos HEX solamente pueden recibir dígitos hexadecimales.

### Respuesta del usuario

Asegúrese de que el origen de la asignación incluye solamente dígitos hexadecimales.

---

## Código de error de ejecución de Java EGL VGJ0110E

VGJ0110E: Se ha comparado el elemento HEX %1 con el valor no hexadecimal: %2.

### Descripción

Los elementos HEX solamente pueden compararse con dígitos hexadecimales.

### Respuesta del usuario

Asegúrese de que el valor de comparación incluye solamente dígitos hexadecimales.

---

## Código de error de ejecución de Java EGL VGJ0111E

VGJ0111E: Se ha comparado el elemento HEX %1 con el valor no hexadecimal de %2: %3.

### Descripción

Los elementos HEX solamente pueden compararse con dígitos hexadecimales.

### Respuesta del usuario

Asegúrese de que el valor de comparación contiene solamente dígitos hexadecimales.

---

## Código de error de ejecución de Java EGL VGJ0112E

VGJ0112E: Se ha asignado al elemento NUM %1 el valor no numérico: %2.

### Descripción

Solamente pueden asignarse valores numéricos a los elementos NUM. Dichos valores contienen dígitos y pueden tener espacios iniciales y de cola, una coma decimal y un signo inicial. La coma decimal está permitida entre dos dígitos, justo antes del primer dígito o justo después del último dígito.

### Respuesta del usuario

Asegúrese de que el valor origen es numérico.

---

## Código de error de ejecución de Java EGL VGJ0113E

VGJ0113E: Se ha asignado al elemento NUM %1 el valor no numérico de %2: %3.

### Descripción

Solamente pueden asignarse valores numéricos a los elementos NUM. Dichos valores contienen dígitos y pueden tener espacios iniciales y de cola, una coma decimal y un signo inicial. La coma decimal está permitida entre dos dígitos, justo antes del primer dígito o justo después del último dígito.

### Respuesta del usuario

Asegúrese de que el valor origen es numérico.

---

## Código de error de ejecución de Java EGL VGJ0114E

VGJ0114E: El valor del elemento %1 (%2) no es válido como subíndice.

### Descripción

El valor tiene demasiados dígitos como para ser un subíndice de cualquier elemento de la matriz. Un valor de subíndice debe estar entre uno y el número de apariciones declaradas para el elemento de estructura.

### Respuesta del usuario

Asegúrese de que el valor de índice es un subíndice válido para la matriz.

---

## Código de error de ejecución de Java EGL VGJ0115E

VGJ0115E: No se puede asignar una serie a %1. La serie es %2.

### Descripción

No se puede asignar una serie al elemento.

### Respuesta del usuario

No asigne una serie al elemento.

---

## Código de error de ejecución de Java EGL VGJ0116E

VGJ0116E: No se puede asignar un número a %1. El número es %2.

### Descripción

No se puede asignar un número al elemento.

### Respuesta del usuario

No asigne un número al elemento.

---

## Código de error de ejecución de Java EGL VGJ0117E

VGJ0117E: No puede convertirse %1 a largo.

### Descripción

No puede convertirse el elemento a largo.

### Respuesta del usuario

Complete los pasos siguientes:

1. Registre el número de mensaje y el texto del mensaje.

**Nota:** El mensaje de error incluye la siguiente información de importancia:

- Dónde se produjo el error
  - El tipo de error interno
2. Registre la situación en la que aparece este mensaje.
  3. Para obtener más instrucciones sobre cómo informar de posibles defectos al Centro de soporte de IBM, consulte la *Guía de instalación de EGL*.
- 

## Código de error de ejecución de Java EGL VGJ0118E

VGJ0118E: %1 no puede convertirse a un número.

### Descripción

El elemento no puede convertirse a un número.

### Respuesta del usuario

No utilizar el elemento en un lugar en el que se necesite un número.

---

## Código de error de ejecución de Java EGL VGJ0119E

VGJ0119E: %1 no es un número válido.

### Descripción

Mientras utilizaba el depurador, el usuario ha intentado establecer el valor de un elemento numérico, pero el nuevo valor no es un número.

### Respuesta del usuario

Utilice un valor numérico.

---

## Código de error de ejecución de Java EGL VGJ0120E

VGJ0120E: %1 no es un valor válido para el índice inicial del operador de subserie en el elemento %2.

### Descripción

El índice inicial no puede ser menor que 1 ni mayor que la longitud del elemento.

### Respuesta del usuario

Utilice un índice válido para el índice inicial del operador de subserie.

---

## Código de error de ejecución de Java EGL VGJ0121E

VGJ0121E: %1 no es un valor válido para el índice final del operador de subserie en el elemento %2.

### Descripción

El índice final no puede ser menor que 1 ni mayor que la longitud del elemento.

### Respuesta del usuario

Utilice un índice válido para el índice final del operador de subserie.

---

## Código de error de ejecución de Java EGL VGJ0122E

VGJ0122E: El índice final del operador de subserie en el elemento %1 es %2, que no puede ser menor que el índice inicial, que es %3.

### Descripción

El índice final del operador de subserie no puede ser menor que el índice inicial.

### Respuesta del usuario

Asegúrese de que el índice inicial sea menor o igual que el índice final.

---

## Código de error de ejecución de Java EGL VGJ0123E

VGJ0123E: El operador de subserie ha fallado: %1 no puede utilizarse como un valor de serie.

### Descripción

La variable no soporta el operador de subserie.

### Respuesta del usuario

Cambie el programa de forma que no utilice el operador de subserie en la variable.

---

## Código de error de ejecución de Java EGL VGJ0124E

VGJ0124E: No es posible asignar un registro a %1. El registro era %2.

### Descripción

El tipo del elemento de datos no permite las asignaciones de registros.

### Respuesta del usuario

Cambie el programa de forma que no asigne un registro al elemento de datos.

---

## Código de error de ejecución de Java EGL VGJ0125E

VGJ0125E: %1 no puede utilizarse como un campo.

### Descripción

La variable no es un campo.

### Respuesta del usuario

Cambie el programa de forma que no utilice la variable como un campo.

---

## Código de error de ejecución de Java EGL VGJ0126E

VGJ0126E: tipos incompatibles en comparación de %1 a %2.

### Descripción

Los tipos de valores son incompatibles en una comparación.

### Respuesta del usuario

Asegúrese de que la comparación utilice tipos compatibles.

---

## Código de error de ejecución de Java EGL VGJ0127E

VGJ0127E: No es posible asignar un valor de fecha u hora a %1. El valor era %2.

### Descripción

No es posible asignar una valor de fecha u hora al elemento.

### Respuesta del usuario

No asigne un valor de fecha u hora al elemento.

---

## Código de error de ejecución de Java EGL VGJ0140E

VGJ0140E: La función de matriz %1 ha fallado porque se ha intentado expandir la matriz %2 por encima de su tamaño máximo.

### Descripción

La matriz no puede contener más valores.

### Respuesta del usuario

Modifique el programa para comprobar el tamaño de la matriz antes de intentar añadirle valores.

---

## Código de error de ejecución de Java EGL VGJ0141E

VGJ0141E: %1 es un índice no válido para la matriz %2. Tamaño actual: %3. Tamaño máximo: %4

### Descripción

El índice está fuera de rango para la matriz.

### Respuesta del usuario

Modifique el programa para que utilice un índice de matriz válido.

---

## Código de error de ejecución de Java EGL VGJ0142E

VGJ0142E: No puede cambiarse el `maximumSize` de la matriz %1. Se esperaba %2, se obtuvo %3.

### Descripción

La matriz se ha pasado en una sentencia `call`. La matriz correspondiente en el programa llamado tiene un `maximumSize` distinto.

### Respuesta del usuario

Modifique uno de los programas de forma que ambos utilicen una matriz con el mismo `maximumSize`.

---

## Código de error de ejecución de Java EGL VGJ0143E

VGJ0143E: %1 no es un tamaño válido para la matriz %2.

### Descripción

La matriz se ha pasado en una sentencia `call`. El programa llamado ha cambiado el tamaño de la matriz a un valor menor que cero o mayor que el valor de la propiedad `maxSize`.

### Respuesta del usuario

Modifique los programas para que utilicen el mismo valor para la propiedad `maxSize`.

---

## Código de error de ejecución de Java EGL VGJ0144E

VGJ0144E: %1 ha fallado para la matriz %2. Se han especificado demasiados tamaños.

### Descripción

La función especificada ha fallado. Este argumento es una matriz de tamaños que contenía demasiados elementos.

### Respuesta del usuario

Corrija el argumento.

---

## Código de error de ejecución de Java EGL VGJ0145E

VGJ0145E: %1 ha fallado para la matriz %2. Los tamaños deben ser elementos de datos numéricos.

### Descripción

La función especificada ha fallado. El argumento debería haber sido una matriz de elementos de datos numéricos, pero no lo era.

### Respuesta del usuario

Corrija el argumento.

---

## Código de error de ejecución de Java EGL VGJ0146E

VGJ0146E: %1 ha fallado para la matriz %2. El tamaño dado era menor que cero.

### Descripción

La función especificada ha fallado. Este argumento es una matriz de tamaños. Uno de los tamaños era menor que cero, pero esto no está permitido.

### Respuesta del usuario

Corrija el argumento.

---

## Código de error de ejecución de Java EGL VGJ0147E

VGJ0147E: %1 ha fallado para la matriz %2. El `maxSize` dado es menor que el tamaño actual.

### Descripción

El `maxSize` de la matriz no puede cambiarse por un valor menor que el tamaño actual.

### Respuesta del usuario

Corrija el argumento.

---

## Código de error de ejecución de Java EGL VGJ0160E

VGJ0160E: La función matemática %1 ha fallado con el código de error 8 (error de dominio).

### Descripción

Un argumento de la función no es válido.

### Respuesta del usuario

Haga lo siguiente:

- Modifique la lógica del programa para asegurarse de que los argumentos de la función son válidos, según la documentación de la función; o bien
- Llame a la función en una sentencia try o establezca `VGVar.handleSysLibraryErrors` en 1 antes de llamar a la función, de forma que el programa pueda manejar el error.

---

## Código de error de ejecución de Java EGL VGJ0161E

VGJ0161E: La función matemática %1 ha fallado con el código de error 8 (error de dominio).

### Descripción

El argumento debe estar entre -1 y 1.

### Respuesta del usuario

Haga lo siguiente:

- Modifique la lógica del programa para asegurarse de que el argumento pasado a la función está entre -1 y 1; o bien
- Llame a la función en una sentencia try o establezca `VGVar.handleSysLibraryErrors` en 1 antes de llamar a la función, de forma que el programa pueda manejar el error.

---

## Código de error de ejecución de Java EGL VGJ0162E

VGJ0162E: La función matemática atan2 ha fallado con el código de error 8 (error de dominio).

### Descripción

Ambos argumentos no pueden ser cero.

### Respuesta del usuario

Haga lo siguiente:

- Modifique la lógica del programa para asegurarse de que al menos un argumento pasado a la función no es cero; o bien

- Llame a la función en una sentencia try o establezca `VGVar.handleSysLibraryErrors` en 1 antes de llamar a la función, de forma que el programa pueda manejar el error.

---

## Código de error de ejecución de Java EGL VGJ0163E

**VGJ0163E: La función matemática %1 ha fallado con el código de error 8 (error de dominio).**

### Descripción

El segundo argumento no debe ser cero.

### Respuesta del usuario

Haga lo siguiente:

- Modifique la lógica del programa para asegurarse de que el segundo argumento no es cero; o bien
- Llame a la función en una sentencia try o establezca `VGVar.handleSysLibraryErrors` en 1 antes de llamar a la función, de forma que el programa pueda manejar el error.

---

## Código de error de ejecución de Java EGL VGJ0164E

**VGJ0164E: La función matemática %1 ha fallado con el código de error 8 (error de dominio).**

### Descripción

El argumento debe ser mayor que cero.

### Respuesta del usuario

Haga lo siguiente:

- Modifique la lógica del programa para asegurarse de que el argumento pasado a la función es mayor que cero; o bien
- Llame a la función en una sentencia try o establezca `VGVar.handleSysLibraryErrors` en 1 antes de llamar a la función, de forma que el programa pueda manejar el error.

---

## Código de error de ejecución de Java EGL VGJ0165E

**VGJ0165E: La función matemática pow ha fallado con el código de error 8 (error de dominio).**

### Descripción

Si el primer argumento es cero, el segundo debe ser mayor que cero.

### Respuesta del usuario

Haga lo siguiente:

- Modifique la lógica del programa para asegurarse de que si el primer argumento pasado a la función es cero, el segundo argumento es mayor que cero; o bien

- Llame a la función en una sentencia try o establezca `VGVar.handleSysLibraryErrors` en 1 antes de llamar a la función, de forma que el programa pueda manejar el error.

---

## Código de error de ejecución de Java EGL VGJ0166E

**VGJ0166E:** La función matemática `pow` ha fallado con el código de error 8 (error de dominio).

### Descripción

Si el primer argumento es menor que cero, el segundo debe ser un entero.

### Respuesta del usuario

Haga lo siguiente:

- Modifique la lógica del programa para asegurarse de que si el primer argumento pasado a la función es menor que cero, el segundo argumento es un entero; o bien
- Llame a la función en una sentencia try o establezca `VGVar.handleSysLibraryErrors` en 1 antes de llamar a la función, de forma que el programa pueda manejar el error.

---

## Código de error de ejecución de Java EGL VGJ0167E

**VGJ0167E:** La función matemática `sqrt` ha fallado con el código de error 8 (error de dominio).

### Descripción

El argumento debe ser mayor que cero o igual.

### Respuesta del usuario

Haga lo siguiente:

- Modifique la lógica del programa para asegurarse de que el argumento pasado a la función es mayor que cero o igual; o bien
- Llame a la función en una sentencia try o establezca `VGVar.handleSysLibraryErrors` en 1 antes de llamar a la función, de forma que el programa pueda manejar el error.

---

## Código de error de ejecución de Java EGL VGJ0168E

**VGJ0168E:** La función matemática `%1` ha fallado con el código de error 12 (error de rango).

### Descripción

Un resultado intermedio o final no puede representarse como un número de coma flotante de precisión doble o con la precisión del elemento de resultado.

### Respuesta del usuario

Haga lo siguiente:

- Modifique la lógica del programa para asegurarse de que el elemento destino es lo suficientemente grande como para contener el valor del resultado; o bien
- Modifique la lógica del programa para asegurarse de que los argumentos para la función tienen valores que no provocan este problema; o bien
- Llame a la función en una sentencia try o establezca `VGVar.handleSysLibraryErrors` en 1 antes de llamar a la función, de forma que el programa pueda manejar el error.

---

## Código de error de ejecución de Java EGL VGJ0200E

VGJ0200E: La función de serie %1 ha fallado con el código de error 8.

### Descripción

El índice debe estar entre 1 y la longitud de la serie.

### Respuesta del usuario

Haga lo siguiente:

- Modifique la lógica del programa para asegurarse de que el argumento para la función relacionado con el índice está entre 1 y la longitud de la serie; o bien
- Llame a la función en una sentencia try o establezca `VGVar.handleSysLibraryErrors` en 1 antes de llamar a la función, de forma que el programa pueda manejar el error.

---

## Código de error de ejecución de Java EGL VGJ0201E

VGJ0201E: La función de serie %1 ha fallado con el código de error 12.

### Descripción

La longitud debe ser mayor que cero.

### Respuesta del usuario

Haga lo siguiente:

- Modifique la lógica del programa para asegurarse de que los argumentos de longitud pasados a la función tienen valores mayores que cero; o bien
- Llame a la función en una sentencia try o establezca `VGVar.handleSysLibraryErrors` en 1 antes de llamar a la función, de forma que el programa pueda manejar el error.

---

## Código de error de ejecución de Java EGL VGJ0202E

VGJ0202E: La función de serie `setNullTerminator` ha fallado con el código de error 16.

### Descripción

El último byte de la serie destino debe ser un blanco o un carácter nulo.

### Respuesta del usuario

Haga lo siguiente:

- Modifique la lógica del programa para asegurarse de que el último byte de la serie destino es un blanco o un carácter nulo; o bien
- Llame a la función en una sentencia try o establezca `VGVar.handleSysLibraryErrors` en 1 antes de llamar a la función, de forma que el programa pueda manejar el error.

---

## Código de error de ejecución de Java EGL VGJ0203E

VGJ0203E: La función de serie %1 ha fallado con el código de error 20.

### Descripción

El índice de una subserie DBCHAR o UNICODE debe ser impar para que el índice identifique el primer byte de un carácter.

### Respuesta del usuario

Haga lo siguiente:

- Modifique la lógica del programa para asegurarse de que los argumentos de índice pasados a la función son válidos; o bien
- Llame a la función en una sentencia try o establezca `VGVar.handleSysLibraryErrors` en 1 antes de llamar a la función, de forma que el programa pueda manejar el error.

---

## Código de error de ejecución de Java EGL VGJ0204E

VGJ0204E: La función de serie %1 ha fallado con el código de error 24.

### Descripción

La longitud de una subserie DBCHAR o UNICODE debe ser par para hacer referencia a un número entero de caracteres.

### Respuesta del usuario

Haga lo siguiente:

- Modifique la lógica del programa para asegurarse de que los argumentos de longitud pasados a la función tienen valores válidos; o bien
- Llame a la función en una sentencia try o establezca `VGVar.handleSysLibraryErrors` en 1 antes de llamar a la función, de forma que el programa pueda manejar el error.

---

## Código de error de ejecución de Java EGL VGJ0215E

VGJ0215E: Se ha pasado a %1 la serie no numérica %2.

### Descripción

Cada carácter de la parte de la serie definida por el argumento de longitud debe ser numérico.

## Respuesta del usuario

Modifique la lógica del programa de forma que los caracteres de la parte de la serie definida por el argumento de longitud sean numéricos.

---

## Código de error de ejecución de Java EGL VGJ0216E

VGJ0216E: %1 no es una máscara de fecha válida para %2.

### Descripción

La máscara de fecha definida en el archivo de propiedades para su uso con la función no es válida.

Los caracteres válidos para una máscara de fecha son los siguientes:

#### D, M, Y

D para Día, M para Mes, Y para Año

#### Carácter separador

Cualquier carácter no numérico de un solo byte, excepto D, M o Y.

Las máscaras de fecha válidas pueden tener cualquiera de los siguientes formatos:

- Gregoriana larga

La versión larga de la máscara Gregoriana debe contener los siguientes componentes en cualquier orden:

**YYYY** Año de 4 dígitos

**MM** Mes numérico de 2 dígitos

**DD** Día del mes numérico de 2 dígitos

Los componentes de máscara deben estar separados por cualquier carácter no numérico de un solo byte excepto D, M o Y.

Por ejemplo, una máscara de YYYY/MM/DD se utiliza para visualizar el 25 de agosto de 1997 como 1997/08/25.

- Juliana larga

La versión larga de la máscara Juliana debe contener los siguientes componentes en cualquier orden:

**YYYY** Año de 4 dígitos

**DDD** Día del año numérico de 3 dígitos

Los componentes de máscara deben estar separados por cualquier carácter no numérico de un solo byte excepto D, M o Y.

Por ejemplo, una máscara de DDD-YYYY puede utilizarse para visualizar el 25 de agosto de 1997 como 237-1997.

## Respuesta del usuario

Cambie la máscara de fecha correctamente a un valor válido y reinicie el programa. Si no se ha definido ninguna propiedad de máscara de fecha, se utilizará una máscara de fecha por omisión.

Las máscaras de fecha pueden establecerse utilizando las propiedades `vgj.datemask.gregorian.long.NNN` y `vgj.datemask.julian.long.NNN`, donde `NNN` es el código NLS actual.

---

## Código de error de ejecución de Java EGL VGJ0217E

VGJ0217E: Se ha producido un error en la función de conversión con argumento %1: %2

### Descripción

El intento de convertir los datos del argumento ha fallado. El motivo de la anomalía está incluido en el mensaje.

### Respuesta del usuario

Utilice el mensaje de error para diagnosticar y corregir el problema.

---

## Código de error de ejecución de Java EGL VGJ0218E

VGJ0218E: GetMessage ha fallado. No se ha podido encontrar el mensaje para la clave %1.

### Descripción

No se ha encontrado ningún mensaje para la clave pasada a la función del sistema getMessage.

### Respuesta del usuario

Añada el mensaje o utilice una clave distinta.

---

## Código de error de ejecución de Java EGL VGJ0250E

VGJ0250E: No se ha podido recuperar el elemento %1 del componente continente %2.

### Descripción

Se ha producido un error interno. Se ha intentado acceder a un elemento con el índice especificado en el registro o tabla.

### Respuesta del usuario

Haga lo siguiente:

1. Registre el número de mensaje y el texto del mensaje.

**Nota:** El mensaje de error incluye la siguiente información de importancia:

- Dónde se produjo el error
  - El tipo de error interno
2. Registre la situación en la que aparece este mensaje.
  3. Para obtener más instrucciones sobre cómo informar de posibles defectos al Centro de soporte de IBM, consulte la *Guía de instalación de EGL*.

---

## Código de error de ejecución de Java EGL VGJ0300E

VGJ0300E: No se ha podido cargar el archivo de tabla para la tabla %1. No se ha encontrado un archivo denominado %2 o %3.

## Descripción

No se ha encontrado ninguno de los archivos en ninguna de las ubicaciones de recursos. Se busca el primer archivo en todas las ubicaciones de recursos. Si no existe tal archivo, se busca el segundo archivo en todas las ubicaciones de recursos.

Las ubicaciones de recursos difieren dependiendo del mecanismo utilizado para localizar el archivo de tabla.

Si el error se ha encontrado en un applet, las ubicaciones de recursos hacen referencia a ubicaciones en la máquina servidor y pueden variar dependiendo de la implementación de la Máquina virtual Java. No obstante, todas las implementaciones deberán buscar en el directorio en el servidor especificado por el valor CODEBASE. Este valor lo establece el código APPLET en el archivo HTML que contiene el applet. Si no se ha especificado ningún valor CODEBASE, toma por omisión el directorio del servidor web que contenga el archivo HTML.

Si el error se ha encontrado en una aplicación, las ubicaciones de recursos válidas son las siguientes:

- El directorio en que se ha iniciado la máquina virtual Java (el directorio de trabajo para el ejecutable).
- Cualquier directorio especificado en la CLASSPATH para la aplicación que se está ejecutando. La especificación de este valor depende del sistema. En algunos sistemas, puede especificarse como una variable de entorno. Todos los sistemas permiten que se especifique al invocar a la máquina virtual Java utilizando la opción `-classpath`. Consulte la documentación incluida con su copia de la Máquina virtual Java para obtener información sobre el valor de CLASSPATH.

## Respuesta del usuario

Primero localice el archivo de tabla y asegúrese de que se han establecido los permisos necesarios para acceder a él.

Si el error se ha producido desde dentro de un applet o si el error se ha producido desde dentro de una aplicación y no desea modificar el conjunto de ubicaciones de recursos existente, copie el archivo de tabla a una ubicación de recursos válida.

De lo contrario, complete una de las siguientes instrucciones:

- Si el intérprete de Java va a utilizar el valor de la variable de entorno CLASSPATH, añada el directorio que contiene el archivo de tabla al valor actual de CLASSPATH.
- Especifique el directorio que contiene el archivo de tabla utilizando la opción `-classpath` al invocar al intérprete de Java. Si especificar la opción `-classpath` altera temporalmente el valor de la variable de entorno CLASSPATH, deberá especificar la vía de acceso a las clases de ejecución de Java (por ejemplo, `classes.zip` o `rt.jar`) además de los directorios que añada como ubicaciones de recursos.

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

---

## Código de error de ejecución de Java EGL VGJ0301E

VGJ0301E: No se ha podido cargar el archivo de tabla %1 para la tabla %2 porque se ha devuelto un número de bytes incorrecto durante la operación de lectura en la cabecera de tabla.

### Descripción

Existe una de las siguientes condiciones:

- El archivo de tabla se ha corrompido.
- El archivo de tabla no se ha generado con EGL o VisualAge Generator.

### Respuesta del usuario

Vuelva a generar la tabla.

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

Si el problema persiste, haga lo siguiente:

1. Registre el número de mensaje y el texto del mensaje.

**Nota:** El mensaje de error incluye la siguiente información de importancia:

- Dónde se produjo el error
  - El tipo de error interno
2. Registre la situación en la que aparece este mensaje.
  3. Para obtener más instrucciones sobre cómo informar de posibles defectos al Centro de soporte de IBM, consulte la *Guía de instalación de EGL*.

---

## Código de error de ejecución de Java EGL VGJ0302E

VGJ0302E: No se ha podido cargar el archivo de tabla %1 para la tabla %2 porque se ha encontrado un número mágico inesperado durante la inspección de la cabecera de tabla.

### Descripción

Existe una de las siguientes condiciones:

- El archivo de tabla se ha corrompido.
- El archivo de tabla no se ha generado con EGL o VisualAge Generator.

### Respuesta del usuario

Vuelva a generar la tabla.

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

Si el problema persiste, haga lo siguiente:

1. Registre el número de mensaje y el texto del mensaje.

**Nota:** El mensaje de error incluye la siguiente información de importancia:

- Dónde se produjo el error

- El tipo de error interno
2. Registre la situación en la que aparece este mensaje.
  3. Para obtener más instrucciones sobre cómo informar de posibles defectos al Centro de soporte de IBM, consulte la *Guía de instalación de EGL*.

---

## Código de error de ejecución de Java EGL VGJ0303E

**VGJ0303E: No se ha podido cargar el archivo de tabla %1 para la tabla %2 porque se ha producido un error interno de E/S durante una operación de lectura o cierre.**

### Descripción

Existe una de las siguientes condiciones:

- El archivo de tabla se ha corrompido.
- El archivo de tabla no se ha generado con EGL o VisualAge Generator.

### Respuesta del usuario

Vuelva a generar la tabla.

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

Si el problema persiste, haga lo siguiente:

1. Registre el número de mensaje y el texto del mensaje.

**Nota:** El mensaje de error incluye la siguiente información de importancia:

- Dónde se produjo el error
  - El tipo de error interno
2. Registre la situación en la que aparece este mensaje.
  3. Para obtener más instrucciones sobre cómo informar de posibles defectos al Centro de soporte de IBM, consulte la *Guía de instalación de EGL*.

---

## Código de error de ejecución de Java EGL VGJ0304E

**VGJ0304E: No se ha podido cargar el archivo de tabla %1 para la tabla %2 porque se ha devuelto un número de bytes incorrecto durante la operación de lectura en los datos de la tabla.**

### Descripción

Existe una de las siguientes condiciones:

- El archivo de tabla se ha regenerado tras cambiar sus columnas pero el programa que intenta cargar la tabla no se ha regenerado. Generar solamente la tabla tras cambiar la definición de columnas provoca una incoherencia entre la definición en el archivo de tabla y la definición en el archivo de clase de tabla, que solamente se genera durante la generación del código de ejecución.
- El archivo de tabla se ha corrompido.
- El archivo de tabla no se ha generado con EGL o VisualAge Generator.

## Respuesta del usuario

Haga lo siguiente:

- Si no se ha cambiado la definición de columnas, vuelva a generar la tabla.
- Si se ha cambiado la definición de columnas, elimine el cambio y vuelva a generar la tabla o vuelva a generar el código de ejecución para el programa que utiliza la tabla.

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

Si el problema persiste, haga lo siguiente:

1. Registre el número de mensaje y el texto del mensaje.

**Nota:** El mensaje de error incluye la siguiente información de importancia:

- Dónde se produjo el error
  - El tipo de error interno
2. Registre la situación en la que aparece este mensaje.
  3. Para obtener más instrucciones sobre cómo informar de posibles defectos al Centro de soporte de IBM, consulte la *Guía de instalación de EGL*.

---

## Código de error de ejecución de Java EGL VGJ0305E

**VGJ0305E: No se ha podido cargar el archivo de tabla %1 para la tabla %2. Los datos encontrados en el archivo de tabla para el elemento %3 no están en el formato correcto. El error de formato de datos correspondiente es: %4**

### Descripción

Existe una de las siguientes condiciones:

- El archivo de tabla se ha regenerado tras cambiar sus columnas pero el applet o aplicación que intenta cargar la tabla no se ha regenerado. Generar solamente la tabla tras cambiar la definición de columnas provoca una incoherencia entre la definición en el archivo de tabla y la definición en el archivo de clase de tabla, que solamente se genera durante la generación del código de ejecución.
- El archivo de tabla se ha corrompido.
- El archivo de tabla no se ha generado con Rational Application Developer para z/OS ni con VisualAge Generator.

## Respuesta del usuario

Haga lo siguiente:

- Si no se ha cambiado la definición de columnas, vuelva a generar la tabla.
- Si se ha cambiado la definición de columnas, elimine el cambio y vuelva a generar la tabla o vuelva a generar el código de ejecución para el programa que utiliza la tabla.

Si el problema persiste, haga lo siguiente:

1. Registre el número de mensaje y el texto del mensaje.

**Nota:** El mensaje de error incluye la siguiente información de importancia:

- Dónde se produjo el error

- El tipo de error interno
2. Registre la situación en la que aparece este mensaje.
  3. Para obtener más instrucciones sobre cómo informar de posibles defectos al Centro de soporte de IBM, consulte la *Guía de instalación de EGL*.

---

## Código de error de ejecución de Java EGL VGJ0306E

**VGJ0306E: No se ha podido cargar el archivo de tabla %1 para la tabla %2 porque los datos del archivo de tabla son para un tipo de tabla distinto al de la tabla %2.**

### Descripción

Existe una de las siguientes condiciones:

- El archivo de tabla se ha regenerado tras cambiar sus columnas pero el applet o aplicación que intenta cargar la tabla no se ha regenerado. Generar solamente la tabla tras cambiar la definición de columnas provoca una incoherencia entre la definición en el archivo de tabla y la definición en el archivo de clase de tabla, que solamente se genera durante la generación del código de ejecución.
- El archivo de tabla se ha corrompido.
- El archivo de tabla no se ha generado con EGL o VisualAge Generator.

### Respuesta del usuario

Haga lo siguiente:

- Si no se ha cambiado el tipo de tabla, vuelva a generar la tabla.
- Si se ha cambiado el tipo de tabla, edite la definición de tabla para que sea del tipo correcto y vuelva a generar la tabla o vuelva a generar el código de ejecución para el programa que utiliza la tabla.

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

Si el problema persiste, haga lo siguiente:

1. Registre el número de mensaje y el texto del mensaje.

**Nota:** El mensaje de error incluye la siguiente información de importancia:

- Dónde se produjo el error
  - El tipo de error interno
2. Registre la situación en la que aparece este mensaje.
  3. Para obtener más instrucciones sobre cómo informar de posibles defectos al Centro de soporte de IBM, consulte la *Guía de instalación de EGL*.

---

## Código de error de ejecución de Java EGL VGJ0307E

**VGJ0307E: No se ha podido cargar el archivo de tabla %1 para la tabla %2 porque el archivo de tabla %1 es un archivo de tabla de VisualAge Generator C++ y no está en formato big-endian.**

### Descripción

Los archivos de tabla generados por el generador VisualAge Generator C++ solamente pueden utilizarse con programas Java si la ordenación de bytes utilizada

para codificar datos numéricos dentro de la tabla es big-endian.

### **Respuesta del usuario**

Vuelva a generar la tabla en formato big-endian o como una tabla independiente de la plataforma Java.

Para volver a generar la tabla en formato big-endian, utilice VisualAge Generator para generar la tabla para un sistema destino C++ que sea big-endian (por ejemplo, AIX). Para volver a generar la tabla como una tabla independiente de la plataforma Java, genere la tabla para un sistema destino Java con VisualAge Generator o EGL.

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

---

## **Código de error de ejecución de Java EGL VGJ0308E**

**VGJ0308E: No se ha podido cargar el archivo de tabla %1 para la tabla %2. El archivo de tabla %1 es un archivo de tabla de VisualAge Generator C++ y la codificación de caracteres utilizada en la tabla (%3) no está soportada en el sistema de ejecución.**

### **Descripción**

Los archivos de tabla generados por el generador VisualAge Generator C++ solamente pueden utilizarse con programas Java si el tipo de codificación de caracteres utilizado para los datos dentro de la tabla es el mismo tipo de codificación utilizada por el sistema de ejecución.

### **Respuesta del usuario**

Haga lo siguiente:

1. Determine la codificación de caracteres utilizada en su sistema. Los programas Java utilizan las codificaciones de caracteres ASCII o EBCDIC. La mayoría de estaciones de trabajo utilizan la codificación ASCII. La mayoría de plataformas de sistema principal utilizan la codificación EBCDIC. Si no conoce la codificación utilizada en su sistema, póngase en contacto con el administrador del sistema.
2. Vuelva a generar la tabla utilizando la codificación de caracteres correcta o como una tabla independiente de la plataforma Java.

Para volver a generar la tabla utilizando la codificación de caracteres correcta, utilice VisualAge Generator para generar la tabla para su sistema destino u otro sistema destino C++ que utilice la misma codificación de caracteres. Para volver a generar la tabla como una tabla independiente de la plataforma Java, genere la tabla para un sistema destino Java con VisualAge Generator o EGL.

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

---

## **Código de error de ejecución de Java EGL VGJ0315E**

**VGJ0315E: No se ha encontrado una entrada de tabla compartida para la tabla %1 durante el proceso de descarga de la tabla.**

### Descripción

Se ha producido un error interno.

### Respuesta del usuario

Haga lo siguiente:

1. Registre el número de mensaje y el texto del mensaje.

**Nota:** El mensaje de error incluye la siguiente información de importancia:

- Dónde se produjo el error
  - El tipo de error interno
2. Registre la situación en la que aparece este mensaje.
  3. Para obtener más instrucciones sobre cómo informar de posibles defectos al Centro de soporte de IBM, consulte la *Guía de instalación de EGL*.

---

## Código de error de ejecución de Java EGL VGJ0320E

**VGJ0320E: Ha fallado una rutina de edición en la tabla %1 al comparar la columna de tabla %2 y el campo %3.**

### Descripción

La columna de tabla y el campo tienen tipos que no son válidos para la comparación.

### Respuesta del usuario

Realice una de las siguientes acciones:

- Asegúrese de que los tipos de la columna y el campo son válidos para la comparación haciendo lo siguiente:
  1. Corrija el tipo de la columna o el tipo del campo de forma que la comparación sea válida.
  2. Vuelva a generar el programa.
  3. Ejecute el programa.
- Modifique el programa para que utilice una tabla distinta para la rutina de edición, de forma que la comparación entre la columna y el campo sea válida.

Consulte la salida del rastreo para obtener más información.

---

## Código de error de ejecución de Java EGL VGJ0330E

**VGJ0330E: No se ha encontrado un mensaje con el ID %1 en la tabla de mensajes %2.**

### Descripción

Este error puede producirse durante las siguientes operaciones:

- Búsqueda del valor para el msgField de un formulario.
- Búsqueda del valor con el identificador especificado como un mensaje de edición.

Existe una de las siguientes condiciones:

- No existe un mensaje con este ID en la tabla de mensajes.
- El archivo de tabla o el paquete de recursos de mensajes para la tabla se ha corrompido.

#### **Respuesta del usuario**

Realice una de las siguientes acciones:

- Asegúrese de que existe un mensaje con el ID de mensaje haciendo lo siguiente:
  1. Añada un mensaje a la tabla con el ID de mensaje si no existe todavía.
  2. Vuelva a generar la tabla.
  3. Ejecute el programa.
- Modifique el programa para que utilice un mensaje distinto ya definido en la tabla.
- Modifique el programa para que utilice una tabla de mensajes distinta que contenga un mensaje con el ID de mensaje.

---

### **Código de error de ejecución de Java EGL VGJ0331E**

VGJ0331E: No se ha podido cargar el archivo de tabla de mensajes %1.

#### **Descripción**

No se ha podido cargar la clase para la tabla de mensajes del programa, o no se ha podido crear una instancia de la clase.

#### **Respuesta del usuario**

Asegúrese de que se ha generado la tabla de mensajes.

---

### **Código de error de ejecución de Java EGL VGJ0350E**

VGJ0350E: Se ha producido un error en una llamada al programa %1. El código de error es %2.

#### **Descripción**

Una llamada remota o EJB al programa especificado ha fallado.

#### **Respuesta del usuario**

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

---

### **Código de error de ejecución de Java EGL VGJ0351E**

VGJ0351E: Ha fallado la operación de comprometer: %1

#### **Descripción**

No se han podido comprometer los recursos.

### Respuesta del usuario

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

---

## Código de error de ejecución de Java EGL VGJ0352E

VGJ0352E: Ha fallado la operación de retrotraer: %1

### Descripción

No se han podido retrotraer los recursos.

### Respuesta del usuario

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

---

## Código de error de ejecución de Java EGL VGJ0400E

VGJ0400E: Se ha utilizado un índice de parámetro no válido %1, para la función %2.

### Descripción

Este es un error interno.

### Respuesta del usuario

Póngase en contacto con el soporte de IBM.

---

## Código de error de ejecución de Java EGL VGJ0401E

VGJ0401E: Se ha detectado un descriptor de parámetro no válido para la función %1, parámetro %2.

### Descripción

Este es un error interno.

### Respuesta del usuario

Póngase en contacto con el soporte de IBM.

---

## Código de error de ejecución de Java EGL VGJ0402E

VGJ0402E: El tipo de valor utilizado para el parámetro %1 de la función o programa %2 no es válido.

### Descripción

El valor no puede pasarse como un parámetro, ya que el tipo del valor es incompatible con el tipo del parámetro.

### Respuesta del usuario

Realice una de las siguientes acciones:

- Cambie la definición del parámetro para que coincida con el tipo del valor.
- Cambie el tipo del valor para que coincida con la definición del parámetro.

---

## Código de error de ejecución de Java EGL VGJ0403E

VGJ0403E: Se ha producido un error al ejecutar el script %1. El texto de la excepción es %2.

### Descripción

El script ha provocado que se emita una excepción.

### Respuesta del usuario

Corrija la lógica del programa para evitar el error.

---

## Código de error de ejecución de Java EGL VGJ0416E

VGJ0416E: Se ha producido un error en una llamada al programa %1. El código de error es %2 (%3).

### Descripción

Se ha emitido una excepción durante un intento de ejecutar el programa llamado. El problema puede deberse a una de las siguientes condiciones:

- El programa podría no tener permiso para crear un proceso nuevo.
- El programa llamado podría no existir.
- El programa llamado podría no estar en la vía de acceso del sistema.

### Respuesta del usuario

Haga lo siguiente:

1. Verifique que el programa tiene permiso para crear un proceso nuevo.
2. Verifique que el programa llamado existe.
3. Verifique que el programa llamado está en la vía de acceso del sistema.

Si el problema persiste, haga lo siguiente:

1. Registre el número de mensaje y el texto del mensaje.

**Nota:** El mensaje de error incluye la siguiente información de importancia:

- Dónde se produjo el error
  - El tipo de error interno
2. Registre la situación en la que aparece este mensaje.
  3. Para obtener más instrucciones sobre cómo informar de posibles defectos al Centro de soporte de IBM, consulte la *Guía de instalación de EGL*.

---

## Código de error de ejecución de Java EGL VGJ0450E

VGJ0450E: La operación de E/S %1 con el objeto de E/S %2 ha fallado por este motivo: %3.

### Descripción

Una sentencia de E/S de EGL ha fallado fuera de una sentencia try o cuando el valor de la variable del sistema sysVar.handleHardIoErrors era cero.

### Respuesta del usuario

Revise el mensaje de error y responda como corresponda.

---

## Código de error de ejecución de Java EGL VGJ0500E

VGJ0500E: No se ha recibido entrada para un campo necesario; vuelva a especificar.

### Descripción

No se han tecleado datos en el campo. El campo está definido como necesario.

### Respuesta del usuario

Entre datos en el campo o pulse una tecla de edición de salto para pasar por alto la comprobación de edición. Los blancos no satisfacen el requisito de entrada de datos para ningún tipo de campo. Además, los ceros tampoco satisfacen el requisito de entrada de datos para campos numéricos. El programa continúa.

---

## Código de error de ejecución de Java EGL VGJ0502E

VGJ0502E: Error de tipo de datos en la entrada; vuelva a especificar.

### Descripción

Los datos del campo no son datos numéricos válidos. El campo está definido como numérico.

### Respuesta del usuario

Entre solamente datos numéricos en este campo o pulse una tecla de edición de salto para pasar por alto la comprobación de edición. En cualquiera de las situaciones, el programa continúa.

---

## Código de error de ejecución de Java EGL VGJ0503E

VGJ0503E: Se ha excedido el número de dígitos significativos permitido; vuelva a especificar.

### Descripción

Se han entrado datos en un campo numérico que está definido con posiciones decimales, un signo, un símbolo de moneda o ediciones de separador numérico. Los datos de entrada sobrepasan el número de dígitos significativos que pueden visualizarse dentro de los criterios de edición. El número especificado es

demasiado grande. El número de dígitos significativos no puede sobrepasar la longitud de campo, menos el número de posiciones decimales, menos las posiciones necesarias para caracteres de edición.

#### **Respuesta del usuario**

Entre un número con menos dígitos significativos.

---

### **Código de error de ejecución de Java EGL VGJ0504E**

**VGJ0504E: La entrada no está dentro del rango definido; vuelva a especificar.**

#### **Descripción**

Los datos del campo no están dentro del rango de datos válidos definidos para este elemento.

#### **Respuesta del usuario**

Entre datos que estén dentro del rango definido o pulse una tecla de edición de salto para pasar por alto la comprobación de edición. En cualquiera de los casos, el programa continúa.

---

### **Código de error de ejecución de Java EGL VGJ0505E**

**VGJ0505E: Error de longitud mínima de entrada; vuelva a especificar.**

#### **Descripción**

Los datos del campo no contienen suficientes caracteres para cumplir la longitud mínima necesaria.

#### **Respuesta del usuario**

Entre el número de caracteres necesario para cumplir la longitud mínima o pulse una tecla de edición de salto para pasar por alto la comprobación de edición. En cualquiera de los casos, el programa continúa.

---

### **Código de error de ejecución de Java EGL VGJ0506E**

**VGJ0506E: Error de validez de edición de tabla; vuelva a especificar.**

#### **Descripción**

Los datos del campo no cumplen el requisito de edición de tabla definido para el campo de variable.

#### **Respuesta del usuario**

Entre datos que cumplan el requisito de edición de tabla o pulse una tecla de edición de salto para pasar por alto la comprobación de edición. En cualquiera de los casos, el programa continúa.

---

### **Código de error de ejecución de Java EGL VGJ0507E**

**VGJ0507E: Error de comprobación de módulo en entrada; vuelva a especificar.**

### Descripción

Los datos del campo no cumplen el requisito de comprobación de módulo definido para el campo de variable.

### Respuesta del usuario

Entre datos que se ajusten a la comprobación de módulo definida para el campo de variable pulse una tecla de edición de salto para pasar por alto la comprobación de edición. En cualquiera de los casos, el programa continúa.

---

## Código de error de ejecución de Java EGL VGJ0508E

VGJ0508E: Entrada no válida para el formato de fecha u hora definido %1.

### Descripción

Los datos del campo, definido con una edición de fecha, no cumplen los requisitos de la especificación de formato.

### Respuesta del usuario

Entre la fecha en el formato correcto mostrado en el mensaje.

---

## Código de error de ejecución de Java EGL VGJ0510E

VGJ0510E: Entrada no válida para campo booleano.

### Descripción

El valor tecleado en el campo no se ajusta a la comprobación booleana. La entrada en un campo booleano debe ser 'Y' o 'N' para campos de tipo carácter y 1 o 0 para campos numéricos.

### Respuesta del usuario

Entre 'Y' o 'N' para un campo de tipo carácter o 1 o 0 para un campo numérico, o bien pulse una tecla de edición de salto para pasar por alto la comprobación de edición. En cualquiera de los casos, el programa continúa.

---

## Código de error de ejecución de Java EGL VGJ0511E

VGJ0511E: La tabla de edición %1 no está definida para %2.

### Descripción

Se ha solicitado un mensaje de usuario pero no se ha definido un prefijo de tabla de mensajes de usuario para el programa.

### Respuesta del usuario

Haga que el desarrollador de programas realice una de las siguientes operaciones:

- Añadir el prefijo de tabla de mensajes a la especificación del programa y volver a generar el programa.

- Eliminar el número de mensaje de usuario de la edición de campo y volver a generar.

---

## Código de error de ejecución de Java EGL VGJ0512E

**VGJ0512E: Los datos hexadecimales no son válidos.**

### Descripción

Los datos del campo de variable deben estar en formato hexadecimal. Uno o más de los caracteres que ha entrado no aparece en el siguiente conjunto: a b c d e f A B C D E F 0 1 2 3 4 5 6 7 8 9

### Respuesta del usuario

Entre solamente caracteres hexadecimales en el campo de variable. Los caracteres se alinean por la izquierda y se rellenan con el carácter 0. Los blancos intercalados no están permitidos.

---

## Código de error de ejecución de Java EGL VGJ1234E

**VGJ1234E: El valor entrado no es válido ya que no coincide con el patrón establecido.**

### Descripción

Se ha especificado un valor que no coincide con el patrón

### Respuesta del usuario

Especifique el valor según se especifica en el patrón.

---

## Código de error de ejecución de Java EGL VGJ1234E

**VGJ0514E: Error de longitud máxima de entrada; vuelva a especificar.**

### Descripción

Se ha sobrepasado la longitud máxima establecida para este campo.

### Respuesta del usuario

No sobrepase la longitud máxima especificada.

---

## Código de error de ejecución de Java EGL VGJ0516E

**VGJ0516E: La entrada no está dentro de la lista definida; vuelva a especificar.**

### Descripción

La entrada no está dentro de la lista definida; vuelva a especificar.

### Respuesta del usuario

La entrada no está dentro de la lista definida; vuelva a especificar.

---

## Código de error de ejecución de Java EGL VGJ0517E

VGJ0517E: El formato de fecha/hora especificado %1 no es válido.

### Descripción

El formato fecha/hora especificado no es válido.

### Respuesta del usuario

El formato fecha/hora especificado no es válido.

---

## Código de error de ejecución de Java EGL VGJ0600E

VGJ0600E: No se puede obtener enlace para el programa, %1.

### Descripción

No se encuentra una entrada para el programa especificado en el archivo de propiedades de CSO por uno de los siguientes motivos:

- Se ha especificado un archivo de propiedades incorrecto en la configuración de GatewayServlet.
- La entrada para el programa no se ha especificado en el archivo de propiedades de CSO.
- El archivo de propiedades de CSO no está en el directorio especificado en la configuración de GatewayServlet.

### Respuesta del usuario

Póngase en contacto con el administrador del servidor Web para asegurarse de que se realizan las siguientes operaciones:

- Asegúrese de que la configuración de GatewayServlet especifica el archivo de propiedades de CSO correcto utilizando el parámetro de inicialización linkageTable.
  - Asegúrese de que el programa está definido en el archivo de propiedades de CSO.
- 

## Código de error de ejecución de Java EGL VGJ0601E

VGJ0601E: Se ha producido una excepción al intentar llamar al programa de punto de entrada, %1. Excepción: %2. Mensaje: %3.

### Descripción

Se ha producido un error indefinido al intentar llamar al programa de punto de entrada. La excepción y el mensaje definirán el error con mayor detalle. Una página o un programa de punto de entrada ofrecen al usuario un menú de programas que puede iniciarse utilizando el GatewayServlet.

### Respuesta del usuario

Póngase en contacto con el administrador del servidor Web para asegurarse de que la página de punto de entrada o programa de entrada se especifican correctamente en la configuración de GatewayServlet.

---

## Código de error de ejecución de Java EGL VGJ0603E

VGJ0603E: El bean, %1, no es válido.

### Descripción

El Page Bean o el nombre de bean no son válidos.

### Respuesta del usuario

Póngase en contacto con el administrador de servidor Web para asegurarse de que el nombre de bean es correcto y de que el Page Bean y la Java Server Page se despliegan y se ponen a disponibilidad del GatewayServlet.

---

## Código de error de ejecución de Java EGL VGJ0604E

VGJ0604E: Se ha producido una excepción al intentar cargar el bean, %1.  
Excepción: %2. Mensaje: %3.

### Descripción

Se ha producido un error indefinido al intentar cargar el Page Bean. La excepción y el mensaje definirán el error con mayor detalle.

### Respuesta del usuario

Póngase en contacto con el administrador de servidor Web para asegurarse de que el nombre de bean es correcto y de que el Page Bean y la Java Server Page se despliegan y se ponen a disponibilidad del GatewayServlet.

---

## Código de error de ejecución de Java EGL VGJ0607E

VGJ0607E: Se ha producido una discrepancia de versiones entre el servidor, %1 y el bean, %2.

### Descripción

La versión del Bean de registro de interfaz de usuario no coincide con la versión del Registro de interfaz de usuario utilizada por el programa servidor. Para un correcto funcionamiento, las versiones deben ser compatibles.

### Respuesta del usuario

Póngase en contacto con el desarrollador de programas y genera los beans de registro del programa y de la interfaz de usuario. Póngase en contacto con el administrador del servidor Web para asegurarse de que el bean de registro de interfaz de usuario se despliega en la ubicación correcta.

---

## Código de error de ejecución de Java EGL VGJ0608E

VGJ0608E: Se ha producido un error al intentar definir datos en el bean, %1.  
Excepción: %2. Mensaje: %3.

### Descripción

Se ha producido una excepción al intentar establecer los datos de registro desde la aplicación de servidor al Bean de registro de interfaz de usuario. Se incluyen la excepción y el mensaje como ayuda para determinar el problema.

### Respuesta del usuario

Utilice la excepción y el mensaje incluidos en el mensaje para la determinación de problemas.

---

## Código de error de ejecución de Java EGL VGJ0609I

**VGJ0609I: Se está enlazando una sesión de pasarela para el usuario, %1.**

### Descripción

Este mensaje informativo aparece en el stdout o stderr del servidor de aplicaciones. El mensaje aparece siempre que se crea una sesión web para el usuario.

### Respuesta del usuario

No es necesaria respuesta.

---

## Código de error de ejecución de Java EGL VGJ0610I

**VGJ0610I: Se está finalizando el enlace lógico de una sesión de pasarela para el usuario, %1.**

### Descripción

Este mensaje informativo aparece en el stdout o stderr del servidor de aplicaciones. El mensaje aparece siempre que ha finalizado una sesión web para el usuario. Una sesión finalizará tras un período de inactividad o si se produce un error grave que termina la sesión.

### Respuesta del usuario

No es necesaria respuesta.

---

## Código de error de ejecución de Java EGL VGJ0611E

**VGJ0611E: No se puede establecer una conexión con el SessionIDManager.**

### Descripción

El GatewayServlet no ha podido conectarse al SessionIDManager. El SessionIDManager es el componente que otorga los ID de sesión para los usuarios de pasarela. Se obtiene un ID de sesión para cada sesión activa y el programa servidor lo utiliza para guardar y restaurar datos de aplicaciones.

El SessionIDManager es una aplicación aparte que está a la escucha de conexiones y peticiones de ID. Cuando una sesión finaliza, el SessionIDManager pondrá el ID de sesión a disponibilidad de otras sesiones. El SessionIDManager debe estar activo para poder ejecutar el GatewayServlet.

### Respuesta del usuario

Póngase en contacto con el administrador del servidor Web para iniciar el SessionIDManager. Si ya se ha iniciado, debe establecerse la ubicación del SessionIDManager en la configuración del GatewayServlet.

---

## Código de error de ejecución de Java EGL VGJ0612I

**VGJ0612I:** Se ha conectado una sesión de pasarela al SessionIDManager para el usuario, %1.

### Descripción

Este mensaje informativo aparece en el stdout o stderr del servidor Web. Se ha conectado una sesión al SessionIDManager satisfactoriamente para poder obtener un ID de sesión. El programa de servidor utiliza el ID de sesión para guardar y restaurar datos del programa.

### Respuesta del usuario

No es necesaria respuesta.

---

## Código de error de ejecución de Java EGL VGJ0614E

**VGJ0614E:** Falta un parámetro necesario, %1, en la configuración de GatewayServlet.

### Descripción

No se ha especificado un parámetro necesario en la configuración del servlet. El GatewayServlet no se ejecutará sin estos parámetros.

### Respuesta del usuario

Póngase en contacto con el administrador del servidor Web para asegurarse de que el GatewayServlet está configurado correctamente. Consulte la documentación del servidor de aplicaciones para determinar cómo configurar parámetros de servlet.

---

## Código de error de entorno de ejecución Java EGL VGJ0615E

**VGJ0615E:** La transacción Web %1 no puede ejecutarse en esta instancia del EGL Action Invoker.

### Descripción

Ha habido un problema al crear o recuperar el GatewayRequestHandler para el programa.

### Respuesta del usuario

Asegúrese de que la aplicación se ha generado y desplegado en el servidor.

---

## Código de error de ejecución de Java EGL VGJ0616E

**VGJ0616E:** El parámetro de pasarela %1 no especifica una clase válida: %2

### Descripción

No ha podido cargarse ni crearse una instancia de la clase identificada en la propiedad de pasarela especificada.

### Respuesta del usuario

Asegúrese de que la clase se ha desplegado en el servidor y se ha especificado correctamente en el archivo de propiedades de pasarela.

---

## Código de error de ejecución de Java EGL VGJ0617E

**VGJ0617E: Proporcione información de usuario público válida en el archivo de propiedades de pasarela.**

### Descripción

El nombre de usuario público o la contraseña especificados en el archivo de propiedades de pasarela no es válido.

### Respuesta del usuario

Asegúrese de que los valores de nombre de usuario público y contraseña del archivo de propiedades de pasarela son correctos.

---

## Código de error de ejecución de Java EGL VGJ0700E

**VGJ0700E: Se ha producido un error durante la conexión a base de datos: %1.**

### Descripción

Se ha producido un error durante un intento de conexión a una base de datos. El mensaje de error finaliza con texto del sistema de gestión de la base de datos.

### Respuesta del usuario

Revise el mensaje de error y responda como corresponda. Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

---

## Código de error de ejecución de Java EGL VGJ0701E

**VGJ0701E: Debe establecerse una conexión a base de datos antes de una operación de E/S de SQL.**

### Descripción

Se ha intentado una operación de E/S de SQL antes de establecerse una conexión a base de datos.

### Respuesta del usuario

Una operación de E/S de SQL solamente es válida después de que el programa cree una conexión a base de datos. El programa puede crear una conexión por omisión basada en una propiedad de programa y puede alterar temporalmente el valor por omisión ejecutando la función de conexión del sistema. Revise las páginas de ayuda de EGL para obtener detalles sobre propiedades de programa y

la configuración del acceso a bases de datos.

---

## **Código de error de ejecución de Java EGL VGJ0702E**

**VGJ0702E: Se ha producido un error durante la operación de E/S de SQL %1. %2.**

### **Descripción**

Se ha producido un error durante la operación de E/S de SQL especificada. El mensaje finaliza con texto del sistema de gestión de la base de datos.

### **Respuesta del usuario**

Revise el mensaje y responda como corresponda.

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

---

## **Código de error de ejecución de Java EGL VGJ0703E**

**VGJ0703E: Se ha producido un error durante la configuración de la operación de E/S de SQL %1. %2.**

### **Descripción**

Se ha producido un error durante la configuración de la operación de E/S de SQL especificada.

### **Respuesta del usuario**

Revise el mensaje y responda como corresponda.

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

---

## **Código de error de ejecución de Java EGL VGJ0705E**

**VGJ0705E: Se ha producido un error al desconectar la base de datos %1. %2.**

### **Descripción**

Se ha producido un error durante un intento de desconexión de la base de datos especificada. El mensaje de error finaliza con texto del sistema de gestión de la base de datos.

### **Respuesta del usuario**

Revise el mensaje y responda como corresponda.

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

---

## Código de error de ejecución de Java EGL VGJ0706E

VGJ0706E: No se puede establecer conexión con la base de datos %1. La conexión no existe.

### Descripción

Se ha producido un error durante un intento de establecer la conexión con la base de datos especificada. La conexión solamente puede establecerse con una conexión de base de datos activa dentro de la transacción.

### Respuesta del usuario

Asegúrese de que el nombre de la base de datos coincide con una de las conexiones de base de datos activas establecidas para la transacción.

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

---

## Código de error de ejecución de Java EGL VGJ0707E

VGJ0707E: Se ha producido un error de secuencia de E/S de SQL en %1.

### Descripción

Un error de secuencia puede producirse en estos casos:

- Se produce una sustitución o supresión de EGL pero no precedida de una sentencia setupd o update para el mismo registro de SQL
- Se produce una exploración de EGL pero no precedida de una sentencia setupd o setinq para el mismo registro de SQL

El mensaje identifica la última operación de E/S que el programa ha intentado, ya sea sustituir, suprimir o explorar.

### Respuesta del usuario

Asegúrese de que el orden de las sentencias EGL es correcto.

Puede haber información de diagnóstico adicional disponible si habilita el rastreo de programas.

---

## Código de error de entorno de ejecución Java EGL VGJ0708E

VGJ0708E: Error al cargar las clases de controlador JDBC: %1

### Descripción

Se ha producido un error al cargar las clases del controlador JDBC, que son necesarias para la E/S de SQL.

### Respuesta del usuario

Asegúrese de que las clases del controlador JDBC se han especificado correctamente en la propiedad vgj.jdbc.drivers. Si es necesaria más de una, separe

los nombres con un punto y coma. Asegúrese también de que las clases se encuentran en la vía de acceso de clases.

---

## Código de error de ejecución de Java EGL VGJ0709E

**VGJ0709E:** Una sentencia (%1) ha utilizado una sentencia prepared que no estaba preparada.

### Descripción

La sentencia prepared indicada en el mensaje de error no existe. Las sentencias prepared se crean llamando a la sentencia prepare de EGL.

### Respuesta del usuario

Corrija la lógica del programa añadiendo una prepare antes de utilizar la sentencia prepared.

---

## Código de error de ejecución de Java EGL VGJ0710E

**VGJ0710E:** Una sentencia %1 ha utilizado un conjunto de resultados que está cerrado o no existe.

### Descripción

El conjunto de resultados utilizado por la sentencia no puede utilizarse porque no está abierto o no existe.

### Respuesta del usuario

Corrija la lógica del programa para evitar utilizar conjuntos de resultados no válidos.

---

## Código de error de ejecución de Java EGL VGJ0711E

**VGJ0711E:** Se ha producido un error al conectarse a la base de datos %1: %2

### Descripción

No se ha podido establecer una conexión con la base de datos indicada en el mensaje.

### Respuesta del usuario

Utilice el componente Error de este mensaje para diagnosticar y corregir el problema.

---

## Código de error de ejecución de Java EGL VGJ0712E

**VGJ0712E:** No puede conectarse a la base de datos por omisión. No se ha especificado el nombre de la base de datos por omisión.

### Descripción

No se ha especificado el nombre de la base de datos por omisión, por lo que el programa no puede conectarse a ella.

### Respuesta del usuario

El nombre de la base de datos por omisión puede especificarse de varias maneras. Debe establecerse una de las propiedades `vgj.jdbc.default.database.programName` (donde `programName` es el nombre del programa) y `vgj.jdbc.default.database`. El valor de esa propiedad puede ser el nombre real de la base de datos por omisión, o podría ser el nombre lógico de la base de datos por omisión. Cuando se utiliza un nombre lógico, debe establecerse otra propiedad: `vgj.jdbc.database.logicalName`. El valor de esta propiedad debe ser el nombre real de la base de datos por omisión.

---

## Código de error de ejecución de Java EGL VGJ0713E

**VGJ0713E: GET ha fallado porque el conjunto de resultados %1 no se ha abierto con desplazamiento.**

### Descripción

Solo se permite GET NEXT cuando la sentencia OPEN no especifica SCROLL.

### Respuesta del usuario

Añada la opción de desplazamiento a la sentencia de apertura cuando se cree el conjunto de resultados.

---

## Código de error de ejecución de Java EGL VGJ0750E

**VGJ0750E: No se ha podido crear el controlador de E/S para el archivo %1. %2**

### Descripción

Se ha producido una anomalía durante la creación del controlador de E/S para el archivo especificado. Este error puede producirse en los siguientes momentos:

- En la primera operación de E/S para un registro relacionado con el archivo especificado; o bien
- En el primer acceso a la variable del sistema `resourceAssociation` para un registro relacionado con el archivo especificado.

El final del mensaje indica el motivo de la anomalía.

### Respuesta del usuario

Revise el mensaje de error y responda como corresponda.

---

## Código de error de ejecución de Java EGL VGJ0751E

**VGJ0751E: No se ha encontrado la propiedad `fileType` para el archivo %1 en la propiedad de ejecución de Java `vgj.ra.fileName.fileType`.**

### Descripción

Debe establecer la siguiente propiedad de ejecución con un tipo de archivo válido:  
`vgj.ra.fileName.fileType`

*fileName*

El nombre del archivo especificado en el mensaje. Este nombre de archivo es un nombre de archivo lógico asociado con un registro EGL.

Para un registro MQ, el valor es mq; para un registro de serie, el valor es seqws. El origen del valor es el componente de asociaciones de recursos en el momento de la generación; específicamente, el elemento de asociación para el archivo, propiedad **fileType**.

### Respuesta del usuario

Haga lo siguiente:

- Añada la propiedad **fileType** de ejecución al archivo de propiedades de ejecución o al descriptor de despliegue; o bien
- Establezca el valor **fileType** en el momento de la generación y vuelva a generar el programa:
  - En el elemento de asociación específico de nombre de archivo del componente de asociaciones de recursos, establezca la propiedad **fileType**
  - En el descriptor de construcción utilizando en la generación, establezca la opción **genProperties** en GLOBAL

Para conocer más detalles, consulte las páginas de ayuda de EGL sobre el elemento de asociación, sobre las propiedades de ejecución de Java y sobre la configuración del entorno.

---

## Código de error de ejecución de Java EGL VGJ0752E

**VGJ0752E: Se ha especificado un fileType %1 no válido para el archivo %2 en el componente de asociaciones de recursos.**

### Descripción

Debe establecer la siguiente propiedad de ejecución con un tipo de archivo válido:

```
vgj.ra.fileName.fileType
```

*fileName*

El nombre del archivo especificado en el mensaje. Este nombre de archivo es un nombre de archivo lógico asociado con un registro EGL.

Para un registro MQ, el valor es mq; para un registro de serie, el valor es seqws. El origen del valor es el componente de asociaciones de recursos en el momento de la generación; específicamente, el elemento de asociación para el archivo, propiedad **fileType**.

### Respuesta del usuario

Haga lo siguiente:

- Cambie la propiedad **fileType** de ejecución en el archivo de propiedades de ejecución o en el descriptor de despliegue; o bien
- Restablezca el valor **fileType** en el momento de la generación y vuelva a generar el programa:
  - En el elemento de asociación específico de nombre de archivo del componente de asociaciones de recursos, cambie la propiedad **fileType**
  - En el descriptor de construcción utilizando en la generación, establezca la opción **genProperties** en GLOBAL

Para conocer más detalles, consulte las páginas de ayuda de EGL sobre el elemento de asociación, sobre las propiedades de ejecución de Java y sobre la configuración del entorno.

---

## Código de error de ejecución de Java EGL VGJ0754E

**VGJ0754E: El elemento de longitud de registro debe contener un valor que divida los datos no de caracteres en los límite del elemento.**

### Descripción

El registro tiene una longitud variable. Cuando se graban sus datos, el elemento de longitud indica cuántos bytes deben grabarse. El último byte de datos debe ser el último byte de un elemento, a menos que el elemento sea un char.

### Respuesta del usuario

Modifique el programa de forma que el valor del elemento de longitud de registro señale al último byte de un elemento, o entre dentro de un elemento char.

---

## Código de error de ejecución de Java EGL VGJ0755E

**VGJ0755E: El valor de occursItem o lengthItem es demasiado grande.**

### Descripción

El registro tiene una longitud variable. Se ha intentado grabar más bytes que los que el registro contiene actualmente.

### Respuesta del usuario

Modifique el programa de forma que el valor de lengthItem o occursItem esté dentro del tamaño del registro.

---

## Código de error de ejecución de Java EGL VGJ0770E

**VGJ0770E: Se ha producido un error al crear el InitialContext o al buscar el entorno java:comp/env. El error es %1**

### Descripción

La excepción se ha emitido desde el constructor de javax.naming.InitialContext, o al invocar el método de búsqueda con el valor "java:comp/env". El programa necesita crear el objeto InitialContext y buscar "java:comp/env" para poder acceder a los valores del entorno J2EE.

### Respuesta del usuario

Utilice el texto de la excepción y la documentación del entorno J2EE para corregir el problema.

---

## Código de error de ejecución de Java EGL VGJ0800E

**VGJ0800E: La asignación de %1 a %2 no es válida.**

### Descripción

Mientras utilizaba el depurador, ha intentado establecer una variable del sistema con un valor no válido.

### Respuesta del usuario

Elija un valor válido, tal como se describe en la página de ayuda para la variable del sistema.

---

## Código de error de ejecución de Java EGL VGJ0801E

VGJ0801E: %1 no puede modificarse o no existe.

### Descripción

Al utilizar el depurador, ha intentado establecer el valor de una variable del sistema que no puede establecerse o que no existe.

### Respuesta del usuario

Revise las páginas de ayuda para obtener una lista de las variables del sistema y para una descripción de cada una de ellas.

---

## Código de error de ejecución de Java EGL VGJ0802E

VGJ0802E: Error al depurar %1: %2

### Descripción

Se ha producido un error al intentar depurar un PageHandler .

### Respuesta del usuario

Utilice el componente Error del mensaje para diagnosticar y corregir el problema.

---

## Código de error de ejecución de Java EGL VGJ0901E

VGJ0901E: El patrón de tramo de fecha/hora (serie de caracteres que declara los componentes de fecha/hora y longitud para un elemento de timeStamp/interval) no es válido.

### Descripción

El patrón de tramo de fecha/hora (serie de caracteres que declara la longitud y los componentes de fecha/hora para un elemento de timeStamp/interval) no es válido.

### Respuesta del usuario

Ajuste el patrón de tramo de fecha/hora adecuadamente.

---

## Código de error de ejecución de Java EGL VGJ0902E

**VGJ0902E:** La precisión del patrón de tramo de fecha/hora (serie de caracteres que declara los componentes de fecha/hora y longitud para un elemento de `timeStamp/interval`) no es válida.

### Descripción

La precisión del patrón de tramo de fecha/hora (serie de caracteres que declara la longitud y los componentes de fecha/hora para un elemento de `timeStamp/interval`) no es válido.

### Respuesta del usuario

Ajuste la precisión del patrón de tramo de fecha/hora en consecuencia.

---

## Código de error de ejecución de Java EGL VGJ0903E

**VGJ0903E:** El código inicial del patrón de tramo de fecha/hora (serie de caracteres que declara los componentes de fecha/hora y longitud para un elemento de `timeStamp/interval`) no es válido.

### Descripción

El código inicial del patrón de tramo de fecha/hora (serie de caracteres que declara los componentes de fecha/hora y longitud para un elemento de `timeStamp/interval`) no es válido.

### Respuesta del usuario

Ajuste el código inicial del patrón de tramo de fecha/hora en consecuencia.

---

## Código de error de ejecución de Java EGL VGJ0904E

**VGJ0904E:** El código final del patrón de tramo de fecha/hora (serie de caracteres que declara los componentes de fecha/hora y longitud para un elemento de `timeStamp/interval`) no es válido.

### Descripción

El código final del patrón de tramo de fecha/hora (serie de caracteres que declara los componentes de fecha/hora y longitud para un elemento de `timeStamp/interval`) no es válido.

### Respuesta del usuario

Ajuste el código final del patrón de tramo de fecha/hora en consecuencia.

---

## Código de error de ejecución de Java EGL VGJ0905E

**VGJ0905E:** O bien el código inicial, o bien el código final del patrón de tramo de fecha/hora (serie de caracteres que declara los componentes de fecha/hora y longitud para un elemento de `timeStamp/interval`) no es válido.

### Descripción

O bien el código inicial, o bien el código final del patrón de tramo de fecha/hora (serie de caracteres que declara los componentes de fecha/hora y longitud para un elemento de timeStamp/interval) no es válido.

### Respuesta del usuario

Ajuste el código inicial o el código final del patrón de tramo de fecha/hora en consecuencia.

---

## Código de error de ejecución de Java EGL VGJ0906E

VGJ0906E: El valor de INTERVAL no es válido.

### Descripción

El valor de INTERVAL no es válido.

### Respuesta del usuario

Ajuste el valor de INTERVAL en consecuencia.

---

## Código de error de ejecución de Java EGL VGJ0907E

VGJ0907E: El valor de TIMESTAMP no es válido.

### Descripción

El valor de TIMESTAMP no es válido.

### Respuesta del usuario

Ajuste el valor de TIMESTAMP en consecuencia.

---

## Código de error de ejecución de Java EGL VGJ0908E

VGJ0908E: El valor de TIME no es válido.

### Descripción

El valor de TIME no es válido.

### Respuesta del usuario

Ajuste el valor de TIME en consecuencia.

---

## Código de error de ejecución de Java EGL VGJ0909E

VGJ0909E: El valor de DATE no es válido.

### Descripción

El valor de DATE no es válido.

### Respuesta del usuario

Ajuste el valor de DATE en consecuencia.

---

## Código de error de ejecución de Java EGL VGJ0910E

VGJ0910E: BLOB o CLOB se ha quedado sin memoria.

### Descripción

BLOB o CLOB se ha quedado sin memoria.

### Respuesta del usuario

Ajuste el tamaño BLOB o CLOB en consecuencia o asícielo al archivo.

---

## Código de error de ejecución de Java EGL VGJ0911E

VGJ0911E: Se ha producido un error interno durante la ejecución de loadTable.  
%1

### Descripción

Se ha producido un error interno durante la ejecución de loadTable.

### Respuesta del usuario

Para conocer la causa del error, consulte el mensaje de error ampliado.

---

## Código de error de ejecución de Java EGL VGJ0912E

VGJ0912E: Se ha producido un error de SQL durante la ejecución de loadTable.  
%1

### Descripción

Se ha producido un error de SQL durante la ejecución de loadTable.

### Respuesta del usuario

Para conocer la causa del error, consulte el mensaje de error ampliado.

---

## Código de error de ejecución de Java EGL VGJ0913E

VGJ0913E: Se ha producido un error de E/S durante la ejecución de loadTable.  
%1

### Descripción

Se ha producido un error de E/S durante la ejecución de loadTable.

### Respuesta del usuario

Para conocer la causa del error, consulte el mensaje de error ampliado.

---

## Código de error de ejecución de Java EGL VGJ0914E

VGJ0914E: Se ha producido un error durante la carga de la biblioteca del sistema VGJSystemCommandProcessing. %1

### Descripción

Se ha producido un error durante la carga de la biblioteca del sistema VGJSystemCommandProcessing.

### Respuesta del usuario

Para conocer la causa del error, consulte el mensaje de error ampliado.

---

## Código de error de ejecución de Java EGL VGJ0915E

VGJ0915E: Se ha producido un error del sistema mientras ejecutaba el mandato del sistema %1. Compruebe en la vía de acceso del sistema si existe el mandato, si es ejecutable, etc.

### Descripción

Se ha producido un error al ejecutar el mandato del sistema.

### Respuesta del usuario

Compruebe en la vía de acceso del sistema si existe el mandato, si es ejecutable, etc.

---

## Código de error de ejecución de Java EGL VGJ0916E

VGJ0916E: Se ha producido un error interno durante la ejecución de loadTable. %1

### Descripción

Se ha producido un error interno durante la ejecución de unloadTable.

### Respuesta del usuario

Para conocer la causa del error, consulte el mensaje de error ampliado.

---

## Código de error de ejecución de Java EGL VGJ0917E

VGJ0917E: Se ha producido un error de SQL durante la ejecución de unloadTable. %1

### Descripción

Se ha producido un error de SQL durante la ejecución de unloadTable.

### Respuesta del usuario

Para conocer la causa del error, consulte el mensaje de error ampliado.

---

## Código de error de ejecución de Java EGL VGJ0918E

VGJ0918E: Se ha producido un error de E/S durante la ejecución de unloadTable.  
%1

### Descripción

Se ha producido un error de E/S durante la ejecución de unloadTable.

### Respuesta del usuario

Para conocer la causa del error, consulte el mensaje de error ampliado.

---

## Código de error de ejecución de Java EGL VGJ0920E

VGJ0920E: Se ha producido un error al devolver %1 de la función nativa C.

### Descripción

El error se ha producido al devolver un valor de C a EGL.

### Respuesta del usuario

Este es un error interno.

---

## Código de error de ejecución de Java EGL VGJ0921E

VGJ0921E: Se ha producido un error al pasar %1 a la función nativa C.

### Descripción

El error ha ocurrido al pasar un valor de EGL a C.

### Respuesta del usuario

Este es un error interno.

---

## Código de error de ejecución de Java EGL VGJ0922E

VGJ0922E: Se ha producido un error al asignar el valor devuelto por la función nativa C a %1.

### Descripción

El error se ha producido al devolver un valor de C a EGL.

### Respuesta del usuario

Este es un error interno.

---

## Código de error de ejecución de Java EGL VGJ0923E

VGJ0923E: Valor demasiado grande para caber en %1.

### Descripción

El número sobrepasa los límites de la variable de recepción smallint o int.

### Respuesta del usuario

Para almacenar números que estén fuera del rango de smallint o int, redefina la variable para utilizar el tipo int o decimal.

---

## Código de error de ejecución de Java EGL VGJ0924E

VGJ0924E: No es posible convertir entre los tipos especificados.

### Descripción

Durante las llamadas de función nativas, EGL intenta cualquier conversión de datos que tenga sentido. Sin embargo, algunas conexiones no están soportadas, como por ejemplo de intervalo a fecha, de indicación de la hora a divisa, etc.

### Respuesta del usuario

Compruebe que ha especificado los tipos de datos que pretendía.

---

## Código de error de ejecución de Java EGL VGJ0925E

VGJ0925E: La pila del argumento está vacía.

### Descripción

Se ha encontrado una excepción de pila vacía al pasar valores a una función nativa C o al devolver valores de ella.

### Respuesta del usuario

Compruebe que el número de variables de recepción no sobrepase el número de valores pasados o devueltos.

---

## Código de error de ejecución de Java EGL VGJ0926E

VGJ0926E: La asignación de memoria ha fallado.

### Descripción

En la llamada de función nativa actual, algo ha necesitado la asignación de memoria, pero no había memoria disponible.

### Respuesta del usuario

Hay varias causas para este error. Por ejemplo, la aplicación solicita más recursos de los que permite la configuración del sistema o un problema con el sistema operativo requiere que reinicie el sistema.

---

## Código de error de ejecución de Java EGL VGJ0927E

VGJ0927E: Calificador de fecha y hora o de intervalo no válido.

### Descripción

Se ha utilizado un calificador no válido al recibir un valor de indicación de la hora o de intervalo en la función C nativa.

### Respuesta del usuario

Compruebe que ha especificado el calificador deseado.

---

## Código de error de ejecución de Java EGL VGJ0928E

**VGJ0928E: La variable de sistema principal de caracteres es demasiado corta para los datos.**

### Descripción

Una variable de sistema principal que no es suficientemente grande se ha utilizado al recibir una serie de caracteres en la función C nativa.

### Respuesta del usuario

Compruebe el tamaño de la variable.

---

## Código de error de ejecución de Java EGL VGJ0929E

**VGJ0929E: La función nativa C, %1, no se ha encontrado.**

### Descripción

La función C especificada no se ha encontrado en la tabla de funciones.

### Respuesta del usuario

Añada una entrada para esta función en la tabla de funciones y vuelva a crear la biblioteca compartida.

---

## Código de error de ejecución de Java EGL VGJ0930E

**VGJ0930E: Una estructura loc\_t se ha modificado inadecuadamente en el código C nativo.**

### Descripción

Se ha pasado un tipo de datos Clob o Blob a una función nativa, pero la estructura loc\_t en la que se ha recibido se ha cambiado inadecuadamente.

### Respuesta del usuario

Compruebe si loc\_loctype, loc\_type o loc\_fname en la estructura loc\_t se han cambiado en el código C nativo.

---

## Código de error de ejecución de Java EGL VGJ0931E

**VGJ0931E: Se ha producido un error al procesar un objeto grande.**

### Descripción

El error se ha producido al realizar una operación interna en un tipo de datos Clob o Blob.

### Respuesta del usuario

Este es un error interno.

---

## Código de error de ejecución de Java EGL VGJ0932E

**VGJ0932E: La función C nativa, %1, no ha devuelto el número de valores correcto esperado por la función llamante.**

### Descripción

Si la función se invocó como parte de una expresión, ha devuelto más de un valor. De lo contrario, el número de variables devueltas difiere del número de variables de recepción.

### Respuesta del usuario

Compruebe que se haya llamado la función correcta. Revise la lógica de la función C nativa, especialmente los valores devueltos por ella, para asegurarse de que siempre devuelva el número de valores esperado.

---

## Código de error de ejecución de Java EGL VGJ0933E

**VGJ0933E: Los intervalos no son compatibles para la operación.**

### Descripción

Algunas combinaciones de valores de intervalo no son significativas y no están permitidas.

### Respuesta del usuario

Revise la compatibilidad de los tipos de datos de intervalo que se pasan o devuelven.

---

## Código de error de ejecución de Java EGL VGJ1000E

**VGJ1000E: %1 ha fallado. Invocara un método o acceder a un campo denominado %2 ha producido un error no manejado. El mensaje de error es %3**

### Descripción

El error se ha producido en una función de acceso a Java. Se ha lanzado una excepción y no se ha llamado a la función dentro de una sentencia try o VGVar.handleSysLibraryErrors es 0, o se ha lanzado algo que no es una excepción, como por ejemplo un error.

### Respuesta del usuario

Utilice la información del mensaje de error para corregir el problema. Si se ha emitido alguna clase de excepción, modifique la lógica del programa para que

maneje el error llamando a la función de acceso a Java dentro de una sentencia try, o estableciendo `VGVar.handleSysLibraryErrors` en 1 antes de invocar la función de acceso a Java.

---

## Código de error de ejecución de Java EGL VGJ1001E

**VGJ1001E: %1 ha fallado. %2 no es un identificador, o bien es el identificador de un objeto nulo.**

### Descripción

El error se ha producido en una función de acceso a Java. No puede utilizarse el identificador porque no hace referencia a un objeto no nulo.

### Respuesta del usuario

Utilice un identificador de un objeto no nulo.

---

## Código de error de ejecución de Java EGL VGJ1002E

**VGJ1002E: %1 ha fallado. Un método público, campo o clase denominado %2 no existe o no puede cargarse, o bien el número o tipos de los parámetros son incorrectos. El mensaje de error es %3**

### Descripción

No se ha encontrado el método, campo o clase utilizados por una función de acceso a Java.

### Respuesta del usuario

Haga lo siguiente:

- Asegúrese de que el objetivo es un método público, campo o clase.
- Asegúrese de que el nombre del método, campo o clase es correcto. Los nombres de clase deben estar calificados con el nombre de su paquete.
- Si el problema es que falta una clase y el nombre es correcto, asegúrese de que el directorio o archivador que contiene la clase está en la vía de acceso de clases de Java.
- Si el problema es que falta un método y el nombre es correcto, asegúrese de que los tipos y número de parámetros son correctos. Compare los valores pasados a la función de acceso a Java con los valores esperados por el método.

---

## Código de error de ejecución de Java EGL VGJ1003E

**VGJ1003E: %1 ha fallado. El tipo de un valor en EGL no coincide con el tipo esperado en Java para %2. EL mensaje de error es %3**

### Descripción

El tipo de un valor pasado a la función de acceso a Java no es correcto.

### Respuesta del usuario

Los valores asignados a campos y los parámetros pasados a métodos y constructores deben tener el tipo correcto. No es necesaria una coincidencia exacta

mientras la conversión entre los tipos sea válida en Java. Por ejemplo, puede utilizarse una subclase en lugar de su superclase, y un tipo primitivo más pequeño, por ejemplo short, en lugar de uno más grande, por ejemplo int.

---

## Código de error de ejecución de Java EGL VGJ1004E

**VGJ1004E: %1 ha fallado. El objetivo es un método que ha devuelto un nulo, un método que no devuelve un valor o un campo cuyo valor es nulo.**

### Descripción

La función de acceso a Java esperaba que el resultado de la operación fuera un objeto no nulo, pero no ha obtenido uno.

### Respuesta del usuario

Para llamar a un método que pueda devolver un nulo o que no devuelva un valor, utilice `javaStore`; o bien utilice la función del sistema Java y no asigne el resultado a un elemento. Para obtener el valor de un campo que puede ser nulo, utilice `javaStoreField`.

---

## Código de error de ejecución de Java EGL VGJ1005E

**VGJ1005E: %1 ha fallado. El valor devuelto no coincide con el tipo del elemento de retorno.**

### Descripción

El valor devuelto por la función de acceso a Java no puede asignarse al elemento de retorno debido a una discrepancia de tipo.

### Respuesta del usuario

Modifique la lógica del programa para que utilice un elemento de retorno de un tipo adecuado.

---

## Código de error de ejecución de Java EGL VGJ1006E

**VGJ1006E: %1 ha fallado. No se ha podido cargar la clase %2 de un argumento convertido temporalmente a nulo. El mensaje de error es %3**

### Descripción

No se ha encontrado la clase del argumento pasado a la función de acceso a Java.

### Respuesta del usuario

Haga lo siguiente:

- Asegúrese de que el nombre de la clase es correcto. Los nombres de clase deben estar calificados con el nombre de un paquete.
- Si el nombre es correcto, asegúrese de que el directorio o archivador que contiene la clase está en la vía de acceso de clases de Java.

---

## Código de error de ejecución de Java EGL VGJ1007E

**VGJ1007E: %1 ha fallado. No se ha podido obtener información sobre el método o campo denominado %2, o se ha intentado establecer el valor de un campo declarado final. El mensaje de error es %3**

### Descripción

Se ha emitido una `SecurityException` o `IllegalAccessException` al intentar obtener información sobre el método o campo, o bien se ha intentado establecer el valor de un campo declarado final. Los campos declarados finales no pueden modificarse.

### Respuesta del usuario

Haga lo siguiente:

- Si el problema se ha producido al establecer un valor, modifique la lógica del programa de forma que el código no intente establecer el valor de un campo declarado final; como alternativa, cambie la declaración del campo.
- Si el problema era el acceso a información, solicite a un administrador del sistema que actualice el archivo de política de seguridad de la Máquina virtual Java de forma que el programa tenga el permiso necesario. El administrador probablemente necesite otorgar `ReflectPermission "suppressAccessChecks"`.

---

## Código de error de ejecución de Java EGL VGJ1008E

**VGJ1008E: %1 ha fallado. %2 es una interfaz o clase abstracta, por lo que no puede llamarse al constructor.**

### Descripción

No se puede llamar al constructor de una interfaz o clase abstracta.

### Respuesta del usuario

Modifique la lógica del programa para llamar al constructor de una clase que no sea abstracta.

---

## Código de error de ejecución de Java EGL VGJ1009E

**VGJ1009E: %1 ha fallado. El método o campo %2 no es estático. Debe utilizarse un identificador en lugar de un nombre de clase.**

### Descripción

Cuando un método o campo no se ha declarado como estático, existe solamente en una instancia específica de una clase, no en la propia clase. En este caso debe utilizarse un identificador del objeto.

### Respuesta del usuario

Modifique la lógica del programa para que utilice un identificador en lugar de un nombre de clase.

---

## Código de error de ejecución de Java EGL VGJ1101W

VGJ1101W: No hay más filas en la dirección que está siguiendo.

### Descripción

El usuario final ha intentado navegar más allá de la última fila.

---

## Código de error de ejecución de Java EGL VGJ1148E

VGJ1148E: El campo de acción "%1" no existe.

### Descripción

La acción OnEvent actual hace referencia a un campo que no puede encontrarse.

### Respuesta del usuario

Verifique la existencia del campo en el formulario actual.

---

## Código de error de ejecución de Java EGL VGJ1149E

VGJ1149E: No se puede insertar otra fila; la matriz de entrada está llena.

### Descripción

La variable utilizada para albergar los datos de la matriz no tiene espacio para otra fila.

### Respuesta del usuario

Aumente el tamaño de almacenamiento de la variable EGL.

---

## Código de error de ejecución de Java EGL VGJ1150E

VGJ1150E: No se ha encontrado la matriz "%1".

### Descripción

No se ha podido encontrar la matriz especificada en ConsoleForm.

### Respuesta del usuario

Verifique que la matriz esté definida correctamente en ConsoleForm y en el programa EGL.

---

## Código de error de ejecución de Java EGL VGJ1151E

VGJ1151E: La variable de resultado de asignación a solicitud ha fallado.

### Descripción

La variable de resultado de asignación a solicitud ha fallado.

### Respuesta del usuario

Verifique que la variable de resultado pueda albergar el resultado de la acción de solicitud.

---

## Código de error de ejecución de Java EGL VGJ1152E

VGJ1152E: El tamaño "%1" del campo de matriz de pantalla no es correcto.

### Descripción

El tamaño del campo de matriz de pantalla especificado no es correcto.

### Respuesta del usuario

Verifique la definición de la matriz de pantalla y la utilización en el programa EGL.

---

## Código de error de ejecución de Java EGL VGJ1153E

VGJ1153E: Los parámetros de DrawBox están fuera de rango.

### Descripción

Los parámetros de DrawBox no caben dentro de las dimensiones de la pantalla/ventana actual

### Respuesta del usuario

Verifique los parámetros de la función drawbox y las dimensiones de la ventana actual.

---

## Código de error de ejecución de Java EGL VGJ1154E

VGJ1154E: Las coordenadas de visualización están fuera de los límites de la ventana.

### Descripción

Las coordenadas de visualización están fuera de los límites de la ventana.

### Respuesta del usuario

Verifique que las coordenadas que se utilizan están dentro del tamaño de la ventana.

---

## Código de error de ejecución de Java EGL VGJ1155E

VGJ1155E: Nombre de clave mal formado "%1".

### Descripción

El nombre de clave especificado no sigue el convenio de denominación de nombres de clave.

### Respuesta del usuario

Reescriba el nombre de clave para que siga los convenios de nombre de clave de EGL.

---

## Código de error de ejecución de Java EGL VGJ1156E

**VGJ1156E:** No puede utilizar esta característica de edición porque ya existe una imagen.

### Descripción

El atributo de imagen restringe las características de edición para este campo.

### Respuesta del usuario

Utilice claves de edición alternativas y acciones para obtener los resultados deseados.

---

## Código de error de ejecución de Java EGL VGJ1157E

**VGJ1157E:** No se puede encontrar la ventana "%1".

### Descripción

No se ha podido ubicar la ventana.

### Respuesta del usuario

Verifique que la ventana se utilice y esté definida adecuadamente.

---

## Código de error de ejecución de Java EGL VGJ1158E

**VGJ1158E:** Los valores nuevos de posición/dimensión de la ventana no son válidos.

### Descripción

Los valores de posición/dimensión especificados no son válidos para el entorno de visualización actual.

### Respuesta del usuario

Verifique que los valores de posición/dimensión sean válidos para el entorno de visualización actual.

---

## Código de error de ejecución de Java EGL VGJ1159E

**VGJ1159E:** La pila de mandatos está desincronizada.

### Descripción

Las sentencias que se ejecutan en las cláusulas OnEvent originan la sincronización de EGL.

### **Respuesta del usuario**

Verifique la utilización de las llamadas de sentencias/función en las sentencias de bloque OnEvent.

---

## **Código de error de ejecución de Java EGL VGJ1160E**

**VGJ1160E: La biblioteca de Interfaz de usuario de consola no está inicializada.**

### **Descripción**

Se ha intentado utilizar la biblioteca de Interfaz de usuario de consola antes de inicializarla.

### **Respuesta del usuario**

Verifique que la secuencia de sentencia de la Interfaz de usuario de consola sea válida.

---

## **Código de error de ejecución de Java EGL VGJ1161E**

**VGJ1161E: Tipo de campo no permitido para la construcción.**

### **Descripción**

El tipo de campo especificado en el campo de consola no es válido para una operación de consulta de construcción.

### **Respuesta del usuario**

Verifique que el tipo de campo del campo de consola sea válido para construir operaciones de consola.

---

## **Código de error de ejecución de Java EGL VGJ1162E**

**VGJ1162E: ConstructQuery no puede llamarse con una lista de variables.**

### **Descripción**

Se ha invocado una operación ConstructQuery con una lista de variables.

### **Respuesta del usuario**

Verifique que la operación de consulta de construcción se haya invocado adecuadamente.

---

## **Código de error de ejecución de Java EGL VGJ1163E**

**VGJ1163E: No se puede inhabilitar un elemento de menú invisible.**

### **Descripción**

El intento de ocultar un elemento de menú invisible no es una operación válida.

### **Respuesta del usuario**

Verifique que el elemento de menú correcto a inhabilitar no sea un elemento de menú invisible.

---

## **Código de error de ejecución de Java EGL VGJ1164E**

**VGJ1164E: La acción de edición ha fallado.**

### **Descripción**

La acción de edición especificada no se ha podido ejecutar.

### **Respuesta del usuario**

Verifique que consolefield está definido adecuadamente y que las acciones de edición que se realicen sean operaciones válidas.

---

## **Código de error de ejecución de Java EGL VGJ1165E**

**VGJ1165E: Se ha producido un error al ejecutar la acción de tecla rápida.**

### **Descripción**

La operación de tecla rápida no ha podido ejecutarse.

### **Respuesta del usuario**

Verifique que la tecla rápida especificada sea válida y que el bloque de sentencia también lo sea.

---

## **Código de error de ejecución de Java EGL VGJ1166E**

**VGJ1166E: No hay ningún mandato activo del que salir.**

### **Descripción**

Se ha hecho un intento de salir del mandato actual, que no existe.

### **Respuesta del usuario**

Verifique que el mandato de salida se esté utilizando en el contexto correcto.

---

## **Código de error de ejecución de Java EGL VGJ1167E**

**VGJ1167E: No hay ningún mandato activo para continuar.**

### **Descripción**

Se ha intentado continuar el mandato actual.

### **Respuesta del usuario**

Verifique que el mandato de continuación se esté utilizando en el contexto correcto.

---

## Código de error de ejecución de Java EGL VGJ1168E

VGJ1168E: Error muy grave: %1

### Descripción

Se ha producido un error de tiempo de ejecución muy grave.

### Respuesta del usuario

Verifique que las sentencias de Interfaz de usuario de consola se estén utilizando en un contexto y una secuencia adecuados.

---

## Código de error de ejecución de Java EGL VGJ1169E

VGJ1169E: El campo "%1" no existe.

### Descripción

El campo de consola especificado no existe.

### Respuesta del usuario

Verifique que el campo de consola se haya definido adecuadamente en el formulario de consola.

---

## Código de error de ejecución de Java EGL VGJ1170E

VGJ1170E: El campo de matriz de pantalla "%1" no es una matriz.

### Descripción

El campo de consola al que se hace referencia en el formulario de consola no es una matriz.

### Respuesta del usuario

Verifique que el campo de consola se defina como una matriz; verifique que se esté haciendo referencia al campo de consola correcto.

---

## Código de error de ejecución de Java EGL VGJ1171E

VGJ1171E: No se ha encontrado el campo "%1".

### Descripción

No se ha podido encontrar el campo de consola especificado.

### Respuesta del usuario

Verifique que el campo de consola se haya definido adecuadamente en el formulario de consola.

---

## Código de error de ejecución de Java EGL VGJ1172E

VGJ1172E: No se puede crear ConsoleField sin una ventana.

### Descripción

Se ha intentado crear un campo de ventana fuera de un contexto de ventana/formulario de consola.

### Respuesta del usuario

Verifique que las definiciones de consoleform y consolefield sean correctas.

---

## Código de error de ejecución de Java EGL VGJ1173E

VGJ1173E: Discrepancia de cuenta de campo de matriz.

### Descripción

El campo de matriz de Interfaz de usuario de consola especificado no coincide con la matriz de EGL especificada.

### Respuesta del usuario

Verifique la definición de matriz y ConsoleField; verifique que se esté utilizando la variable de matriz de EGL correcta en la sentencia openui.

---

## Código de error de ejecución de Java EGL VGJ1174E

VGJ1174E: El formulario "%1" no existe.

### Descripción

El formulario de consola especificado no existe.

### Respuesta del usuario

Verifique que el formulario de consola especificado se defina y se utilice en el contexto correcto.

---

## Código de error de ejecución de Java EGL VGJ1175E

VGJ1175E: El formulario "%1" no cabe en la ventana "%2".

### Descripción

El formulario tiene unas dimensiones que no le permiten ajustarse a las dimensiones de la ventana actual.

### Respuesta del usuario

Altere las dimensiones de la definición de formulario o de la definición de ventana.

---

## Código de error de ejecución de Java EGL VGJ1176E

VGJ1176E: Las listas de campos no coinciden.

### Descripción

La lista de campos especificada no contiene el mismo número de elementos que la lista de variables proporcionada.

### Respuesta del usuario

Altere la sentencia openUI para asegurarse de que se especifica el mismo número de campos y variables.

---

## Código de error de ejecución de Java EGL VGJ1177E

VGJ1177E: El formulario "%1" está ocupado.

### Descripción

La referencia de formulario ya se está utilizando en otro contexto.

### Respuesta del usuario

Verifique que la lógica del programa EGL utiliza un formulario sólo una vez.

---

## Código de error de ejecución de Java EGL VGJ1178E

VGJ1178E: El nombre de formulario "%1" ya se está utilizando.

### Descripción

La definición del formulario ha provocado un conflicto de nombres de formulario.

### Respuesta del usuario

Altere la definición del formulario para que utilice un nombre de formulario exclusivo.

---

## Código de error de ejecución de Java EGL VGJ1179E

VGJ1179E: El formulario "%1" no está abierto.

### Descripción

Se ha intentado hacer una referencia a un objeto de formulario que no está definido.

### Respuesta del usuario

Verifique el formulario especificado está adecuadamente definido y se utiliza en sentencias ConsoleUI válidas.

---

## Código de error de ejecución de Java EGL VGJ1180E

VGJ1180E: No se puede crear ConsoleForm sin una ventana.

### Descripción

Se ha intentado crear ConsoleForm sin una ventana de referencia válida.

### Respuesta del usuario

Verifique que ConsoleForm esté adecuadamente definido y se utilice en una sentencia ConsoleUI válida.

---

## Código de error de ejecución de Java EGL VGJ1181E

VGJ1181E: No se puede utilizar KeyObject.getChar() para teclas virtuales.

### Descripción

consoleUI no puede utilizar KeyObject.getChar() para teclas virtuales.

### Respuesta del usuario

Altere el programa EGL para que construya series para definiciones de teclas virtuales.

---

## Código de error de ejecución de Java EGL VGJ1182E

VGJ1182E: No se puede utilizar KeyObject.getCookedChar() para teclas virtuales.

### Descripción

ConsoleUI no puede utilizar KeyObject.getCookedChar() para teclas virtuales.

### Respuesta del usuario

Altere el programa EGL para que utilice series para definir teclas virtuales.

---

## Código de error de ejecución de Java EGL VGJ1183E

VGJ1183E: La recuperación de la serie de resultado de asignación ha fallado.

### Descripción

La recuperación de la serie de resultado de asignación ha fallado.

### Respuesta del usuario

---

## Código de error de ejecución de Java EGL VGJ1184E

VGJ1184E: La clave de mensaje de ayuda "%1" no se ha encontrado en el paquete compuesto de recursos "%2".

### Descripción

La clave de mensaje de ayuda no se ha podido encontrar en el archivo de ayuda de mensajes especificado.

### Respuesta del usuario

Verifique que se estén utilizando la clave de mensaje de ayuda y el archivo de mensajes de ayuda correctos.

---

## Código de error de ejecución de Java EGL VGJ1185E

VGJ1185E: Suscripción de matriz no permitida.

### Descripción

Se ha intentado hacer referencia a un elemento de matriz no válido.

### Respuesta del usuario

Verifique que la lógica del programa esté haciendo referencia a elementos de matriz dentro del tamaño de la matriz definida.

---

## Código de error de ejecución de Java EGL VGJ1186E

VGJ1186E: No se puede inicializar la biblioteca de Interfaz de usuario de consola.

### Descripción

Al iniciar el programa, no se ha podido inicializar la biblioteca de Interfaz de usuario de consola.

### Respuesta del usuario

Verifique que el programa se esté utilizando en un entorno de pantalla y plataforma soportado.

---

## Código de error de ejecución de Java EGL VGJ1187E

VGJ1187E: ERROR INTERNO

### Descripción

Se ha producido un ERROR INTERNO de ConsoleUI.

### Respuesta del usuario

---

## Código de error de ejecución de Java EGL VGJ1188E

VGJ1188E: Se ha recibido una señal de INTERRUPCIÓN.

### Descripción

Se ha recibido una señal de INTERRUPCIÓN.

### Respuesta del usuario

---

## Código de error de ejecución de Java EGL VGJ1189E

VGJ1189E: No se puede tener un elemento de menú invisible sin acelerador.

### Descripción

Se ha intentado crear un elemento de menú invisible sin tecla aceleradora.

### Respuesta del usuario

Altere la definición del elemento de menú para definir una tecla aceleradora para el elemento de menú invisible.

---

## Código de error de ejecución de Java EGL VGJ1190E

**VGJ1190E: No se puede crear ConsoleLabel sin una ventana.**

### Descripción

Durante la creación de ConsoleLabel, no se ha podido encontrar una referencia de ventana válida.

### Respuesta del usuario

Verifique que la etiqueta de consola esté correctamente definida en el formulario de consola utilizado en el programa EGL.

---

## Código de error de ejecución de Java EGL VGJ1191E

**VGJ1191E: El elemento de menú %1 no cabe en la ventana.**

### Descripción

El elemento de menú especificado es demasiado grande para caber en la ventana activa actual

### Respuesta del usuario

Altere el elemento de menú de forma que el nombre sea más pequeño que la anchura de la ventana activa actual.

---

## Código de error de ejecución de Java EGL VGJ1192E

**VGJ1192E: El elemento de menú "%1" no existe.**

### Descripción

El elemento de menú especificado no se ha podido encontrar o no existe.

### Respuesta del usuario

Verifique que el elemento de menú al que se hace referencia, se haya definido y añadido a la instancia de menú actual.

---

## Código de error de ejecución de Java EGL VGJ1193E

**VGJ1193E: Conflicto de nemotécnicos de menú (tecla=%1).**

### Descripción

Las definiciones de elementos de menú actuales conllevan un conflicto de nemotécnicos.

### Respuesta del usuario

Altere los elementos de menú para asegurarse de que las teclas aceleradoras/OnEvent no entran en conflicto.

---

## Código de error de ejecución de Java EGL VGJ1194E

VGJ1194E: No hay ningún formulario activo.

### Descripción

La Interfaz de usuario de consola no tiene una referencia de formulario activa.

### Respuesta del usuario

Verifique que se haya definido y visualizado un formulario.

---

## Código de error de ejecución de Java EGL VGJ1195E

VGJ1195E: Debe tener un formulario activo para VISUALIZAR MATRIZ.

### Descripción

Se ha intentado visualizar una matriz del formulario activo actual que no existe.

### Respuesta del usuario

Verifique que se haya definido un formulario con una matriz antes de intentar visualizar la matriz.

---

## Código de error de ejecución de Java EGL VGJ1196E

VGJ1196E: Debe tener un formulario activo para LEER MATRIZ.

### Descripción

Se ha intentado leer una matriz del formulario activo que no existe.

### Respuesta del usuario

Verifique que se haya definido un formulario y que se haya activado antes de intentar leer una matriz de él.

---

## Código de error de ejecución de Java EGL VGJ1197E

VGJ1197E: No se puede iniciar un bucle de sucesos sin un mandato actual.

### Descripción

No se puede iniciar un bucle de sucesos sin un mandato actual.

### Respuesta del usuario

---

## Código de error de ejecución de Java EGL VGJ1198E

VGJ1198E: No se ha especificado ningún editor blob.

### Descripción

Se ha intentado editar un blob, pero no se ha especificado ningún editor blob.

### Respuesta del usuario

Defina un editor adecuado en el campo de consola de blob.

---

## **Código de error de ejecución de Java EGL VGJ1199E**

**VGJ1199E: ERROR INTERNO: no hay objeto de formato**

### Descripción

ERROR INTERNO: no hay objeto de formato

### Respuesta del usuario

---

## **Código de error de ejecución de Java EGL VGJ1200E**

**VGJ1200E: No se ha especificado ningún archivo de ayuda.**

### Descripción

Se ha recibido una petición de ayuda, pero no se ha especificado ningún archivo de ayuda.

### Respuesta del usuario

Defina un archivo de ayuda válido en el programa EGL.

---

## **Código de error de ejecución de Java EGL VGJ1201E**

**VGJ1201E: No se ha especificado un mensaje de ayuda.**

### Descripción

Se ha recibido una petición de ayuda, pero no se ha especificado ningún mensaje de ayuda.

### Respuesta del usuario

Altere el programa EGL para proporcionar mensajes de ayuda.

---

## **Código de error de ejecución de Java EGL VGJ1202E**

**VGJ1202E: El menú no está dispuesto.**

### Descripción

Se han intentado utilizar funciones de un menú que no se ha visualizado.

### Respuesta del usuario

Verifique que las funciones del menú se utilizan una vez visualizado éste.

---

## Código de error de ejecución de Java EGL VGJ1203E

VGJ1203E: No hay ninguna matriz de pantalla actual.

### Descripción

Se ha hecho una referencia para utilizar la matriz de pantalla actual que no existe.

### Respuesta del usuario

Verifique que el formulario activo actual contenga una matriz de pantalla.

---

## Código de error de ejecución de Java EGL VGJ1204E

VGJ1204E: No hay elementos de menú visibles.

### Descripción

Durante la construcción de un menú, no se han encontrado elementos de menú visibles.

### Respuesta del usuario

Altere la creación del menú para que al menos haya un elemento de menú visible y visualizable.

---

## Código de error de ejecución de Java EGL VGJ1205E

VGJ1205E: El nombre de la ventana nueva es nulo.

### Descripción

La declaración de la ventana es nula.

### Respuesta del usuario

Proporcione un nombre de ventana al declarar una ventana.

---

## Código de error de ejecución de Java EGL VGJ1206E

VGJ1206E: Se ha intentado abrir una ventana nula.

### Descripción

Se ha intentado abrir una ventana que no existe o está vacía.

### Respuesta del usuario

Verifique que la sentencia de ventana abierta esté utilizando una referencia de ventana válida.

---

## Código de error de ejecución de Java EGL VGJ1207E

VGJ1207E: Se ha producido una excepción en la solicitud.

**Descripción**

Durante la ejecución de una solicitud, se ha producido una excepción.

**Respuesta del usuario**

Verifique que el bloque de sentencia OnEvent de solicitud sea correcto.

---

**Código de error de ejecución de Java EGL VGJ1208E**

VGJ1208E: Se ha recibido una señal QUIT.

**Descripción**

Se ha recibido una señal QUIT.

**Respuesta del usuario****Código de error de ejecución de Java EGL VGJ1209E**

VGJ1209E: No hay ninguna matriz de pantalla activa.

**Descripción**

El formulario activo actual no contiene una matriz de pantalla.

**Respuesta del usuario**

Verifique que la lógica del programa esté utilizando un formulario que contenga una definición de matriz de pantalla.

---

**Código de error de entorno de ejecución Java EGL VGJ1210E**

VGJ1210E: No hay ningún formulario activo.

**Descripción**

La sesión de Interfaz de usuario de consola actual no contiene una instancia de formulario activa.

**Respuesta del usuario**

Verifique que se defina y visualice un formulario antes de que se haga referencia al mismo.

---

**Código de error de entorno de ejecución Java EGL VGJ1211E**

VGJ1211E: No se puede desplazar el menú al elemento actual.

**Descripción**

El intento de mover el cursor de menú a un elemento de menú ha fallado.

### Respuesta del usuario

Verifique que la lógica de menú sea correcta moviendo el cursor de menú al elemento de menú correcto. Verifique que el elemento de menú no esté inhabilitado.

---

## Código de error de ejecución de Java EGL VGJ1212E

VGJ1212E: Atributo desconocido "%1"

### Descripción

El atributo especificado no se ha reconocido.

### Respuesta del usuario

Verifique que el atributo sea correcto para el contexto de Interfaz de usuario de consola actual.

---

## Código de error de ejecución de Java EGL VGJ1213E

VGJ1213E: Error en el campo "%1".

### Descripción

La entrada del campo no es correcta.

### Respuesta del usuario

Verifique que los datos tecleados coincidan con el tipo de datos o las propiedades de formato del campo.

---

## Código de error de ejecución de Java EGL VGJ1214E

VGJ1214E: No se han proporcionado variables suficientes.

### Descripción

La sentencia openUI no se ha proporcionado con variables suficientes para enlazar en el formulario de consola.

### Respuesta del usuario

Altere el programa EGL para que liste más variables para sentencia openUI; altere la sentencia openUI para restringir el número de campos de consola.

---

## Código de error de ejecución de Java EGL VGJ1215E

VGJ1215E: El nombre de ventana "%1" ya se está utilizando.

### Descripción

El nombre de ventana recién definido ya lo está utilizando otra ventana.

### Respuesta del usuario

Altere el nombre de la ventana para que no entre en conflicto con otros nombres de ventana.

---

## Código de error de ejecución de Java EGL VGJ1216E

**VGJ1216E: El tamaño de la ventana es muy pequeño para la pantalla de ayuda.**

### Descripción

Se ha intentado visualizar la pantalla de ayuda en un entorno de visualización que es demasiado pequeño.

### Respuesta del usuario

Ajuste el tamaño del entorno de visualización.

---

## Código de error de ejecución de Java EGL VGJ1217W

**VGJ1217W: No hay más campos en la dirección que está siguiendo.**

### Descripción

Se ha intentado mover el cursor más allá del final de la lista de campos.

### Respuesta del usuario

---

## Código de error de ejecución de Java EGL VGJ1218E

**VGJ1218E: El contenido de la matriz de pantalla '%1' no es válido.**

### Descripción

Un arrayDictionary de la pantalla contiene una entrada para cada columna de la pantalla. Todas las entradas deben del mismo tipo de objeto: ConsoleField o una matriz de ConsoleFields, y todas las matrices (si existen) deben tener el mismo número de elementos.

Examine y corrija la declaración del arrayDictionary de la pantalla.

---

## Código de error de ejecución de Java EGL VGJ1219E

**VGJ1219E: La matriz de pantalla '%1' no puede contener el campo segmentado '%2'.**

### Descripción

La propiedad **segment** de consoleField no debe establecerse en ningún consoleField utilizado en un arrayDictionary de la pantalla.

### Respuesta del usuario

Elimine la propiedad **segment** de los consoleFields incluidos en un arrayDictionary de la pantalla.

---

## Código de error de ejecución de Java EGL VGJ1220E

VGJ1220E: La matriz de pantalla '%1' es incompatible con la matriz de datos.

### Descripción

Un arrayDictionary de la pantalla contiene una entrada para cada columna de la pantalla, y una matriz dinámica está enlazada a esa arrayDictionary. Cada registro de arrayDictionary debe tener el mismo número de campos que el número de columnas del arrayDictionary de la pantalla, y cada campo debe ser compatible por asignación con el consoleField correspondiente.

### Respuesta del usuario

Examine y corrija el arrayDictionary de la pantalla o los registros de la matriz dinámica que está enlazada a dicho arrayDictionary.

---

## Código de error de ejecución de Java EGL VGJ1221E

VGJ1221E: Se han detectado campos con el mismo nombre '%1'.

### Descripción

Un consoleForm no puede tener más de un campo con el mismo nombre.

### Respuesta del usuario

Corrija la declaración de consoleForm.

---

## Código de error de ejecución de Java EGL VGJ1222E

VGJ1222E: La longitud del campo de consola '%1' no es válida.

### Descripción

La longitud de un consoleField debe ser superior a cero.

### Respuesta del usuario

Corrija la declaración de consoleField.

---

## Código de error de ejecución de Java EGL VGJ1223E

VGJ1223E: La etiqueta de [%1, %2] no cabe en el espacio disponible.

### Descripción

Una etiqueta debe caber por completo dentro de los límites de la ventana.

### Respuesta del usuario

Corrija la posición o el tamaño de la etiqueta o de la ventana.

---

## Código de error de ejecución de Java EGL VGJ1224E

VGJ1224E: El segmento del campo '%1' en (%2, %3) no cabe en el espacio disponible.

### Descripción

Un consoleField debe caber por completo dentro de los límites de la ventana.

### Respuesta del usuario

Corrija la posición o el tamaño del consoleField o de la ventana.

---

## Código de error de ejecución de Java EGL VGJ1225E

VGJ1225E: La serie de solicitud es demasiado larga para la ventana activa.

### Descripción

El mensaje que solicita información de entrada del usuario debe caber en la ventana activa. Si la propiedad `isChar` de la solicitud está establecida en `no`, también debe tener en cuenta el espacio necesario para la respuesta del usuario.

### Respuesta del usuario

Corrija la declaración de la solicitud o la ventana.

---

## Código de error de ejecución de Java EGL VGJ1226E

VGJ1226E: Argumentos de matriz OpenUI no válidos.

### Descripción

Un arrayDictionary de la pantalla contiene una entrada para cada columna de la pantalla. Todas las entradas deben del mismo tipo de objeto: ConsoleField o una matriz de ConsoleFields, y todas las matrices (si existen) deben tener el mismo número de elementos.

### Respuesta del usuario

Examine y corrija la declaración del arrayDictionary de la pantalla.

---

## Código de error de ejecución de Java EGL VGJ1227E

VGJ1227E: Argumentos de campo OpenUI no válidos.

### Descripción

En la sentencia `openUI`, puede especificar una lista de consoleFields, especificando consoleFields o especificando series que contengan el valor de los campos de nombre de consoleField. En este caso, la lista incluía un valor no válido.

### Respuesta del usuario

Examine y corrija los valores de la sentencia `openUI`.

---

## Código de error de ejecución de Java EGL VGJ1228E

VGJ1228E: Sólo puede enlazarse una variable a una sentencia prompt.

### Descripción

Una solicitud (prompt) sólo puede estar enlazada a una variable que reciba la respuesta del usuario. Se ha intentado algún otro tipo de enlace.

### Respuesta del usuario

Examine y corrija la sentencia `openUI`.

---

## Código de error de ejecución de Java EGL VGJ1229E

VGJ1229E: No se ha podido determinar el enlace de datos para el campo de consola '%1'.

### Descripción

Si una sentencia `openUI` no enlaza `consoleFields` con otras variables, se utiliza el valor de la propiedad **binding** de cada `consoleField`; pero en este caso no estaba presente la propiedad **binding**.

### Respuesta del usuario

Añada enlaces al `consoleField` o a la sentencia `openUI`.

---

## Código de error de ejecución de Java EGL VGJ1290E

VGJ1290E: %1 no es un parámetro válido para la función Blob/Clob.

### Descripción

Se ha producido un error al procesar una función Blob/Clob. La causa del error está descrita en la inserción de mensaje.

### Respuesta del usuario

Lleve a cabo la acción adecuada basándose en el contenido de la inserción de mensaje.

---

## Código de error de ejecución de Java EGL VGJ1301E

VGJ1301E: Error de cumplimentación de informe %1.

### Descripción

Error de cumplimentación de informe. Los datos proporcionados para el informe no son correctos. Las razones pueden ser que los nombres de campo de registro de matriz dinámica no coincidan con los nombres de campo de informe, que la conexión no exista o que la sentencia SQL no sea válida.

### Respuesta del usuario

Si está utilizando una matriz de registros dinámica, asegúrese de que los nombres de campo definidos en el diseño de informe coincidan por nombre con los elementos del registro. Si está utilizando una sentencia SQL, asegúrese de que SQL sea válido. Si está utilizando una conexión, asegúrese de que la conexión se haya establecido y de que el nombre de la conexión sea correcto. Además, asegúrese de que el nombre de vía de acceso especificado para reportDesignFile sea válido y que el archivo exista.

---

## Código de error de ejecución de Java EGL VGJ1302E

**VGJ1302E: Error de exportación de informe %1.**

### Descripción

Error de exportación de informe. El informe no se ha podido exportar al formato especificado.

### Respuesta del usuario

Asegúrese de que los nombres de vías de acceso sean correctos. El objeto de informe cumplimentado existe en la ubicación especificada y se ha asignado correctamente en el campo reportDestinationFile.

---

## Código de error de ejecución de Java EGL VGJ1303E

**VGJ1303E: Error de acceso dinámico de informe, no se ha encontrado el contenido. %1**

### Descripción

El nombre de campo no existe en la matriz dinámica de registros.

### Respuesta del usuario

Asegúrese de que los nombres de campo coinciden en el diseño de informes y en el registro que está utilizando en el programa EGL.

---

## Código de error de ejecución de Java EGL VGJ1304E

**VGJ1304E: Nombre de conexión incorrecto**

### Descripción

El nombre de conexión no es válido.

### Respuesta del usuario

Asegúrese de que la conexión sea una conexión EGL válida y que la función defineDatabaseAlias se haya utilizado para asignar un nombre a una conexión.

---

## Código de error de ejecución de Java EGL VGJ1305E

**VGJ1305E: La conexión con el nombre especificado %1 no existe**

### Descripción

No existe una conexión con el nombre de conexión.

### Respuesta del usuario

Asegúrese de que las sentencias siguientes estén presentes en el programa de EGL. Una función de conexión con parámetros válidos y un defineDatabaseAlias que da un nombre a la conexión.

---

## Código de error de ejecución de Java EGL VGJ1306E

**VGJ1306E: Correlación de tipos de EGL e Informe incorrecta. Compruebe la tabla de correlaciones.**

### Descripción

Hay una discrepancia de tipos entre los campos del Diseño de informe y los tipos de datos en el programa EGL.

### Respuesta del usuario

Asegúrese de que los tipos sean compatibles tal como se menciona en la documentación. Algunos ejemplos: para un tipo char de EGL, el archivo de diseño debería tener la clase definida como java.lang.String, para un tipo int de EGL, el archivo de diseño debería tener la clase de campo java.lang.Integer.

---

## Código de error de ejecución de Java EGL VGJ1401E

**VGJ1401E: El campo "%1" en la posición(%2,%3) no está dentro del formulario.**

### Descripción

El campo especificado no está dentro del formulario en la posición dada.

### Respuesta del usuario

Verifique que el formulario y los campos estén correctamente definidos.

---

## Código de error de ejecución de Java EGL VGJ1402E

**VGJ1402E: El campo "%1" se solapa con el campo "%2".**

### Descripción

El tamaño y la posición de los dos campos originan un solapamiento.

### Respuesta del usuario

Ajuste las coordenadas de tamaño y posición de los campos.

---

## Código de error de ejecución de Java EGL VGJ1403E

**VGJ1403E: Error interno: no se puede determinar el grupo de formularios.**

### Descripción

Error interno: no se puede determinar el grupo de formularios.

### Respuesta del usuario

Verifique que el formulario y el grupo de formularios estén correctamente definidos.

---

## Código de error de ejecución de Java EGL VGJ1404E

VGJ1404E: El formulario "%1" no cabe en ningún área flotante.

### Descripción

El formulario no cabe en ningún área flotante.

### Respuesta del usuario

Verifique que el formulario pueda visualizarse adecuadamente en un área flotante.

---

## Código de error de ejecución de Java EGL VGJ1405E

VGJ1405E: Las coordenadas del campo "%1" no son válidas.

### Descripción

Las coordenadas del campo no son válidas.

### Respuesta del usuario

Verifique que las coordenadas del campo especificadas sean válidas para el formulario.

---

## Código de error de ejecución de Java EGL VGJ1406E

VGJ1406E: No se puede obtener la asociación de impresión.

### Descripción

El intento de configurar una asociación de impresión ha fallado.

### Respuesta del usuario

Verifique que la asociación de impresora se haya configurado correctamente.

---

## Código de error de ejecución de Java EGL VGJ1407E

VGJ1407E: No existe ningún tamaño de dispositivo de impresión adecuado.

### Descripción

No existe ningún tamaño de dispositivo de impresión adecuado.

## Código de error de ejecución de Java EGL VGJ1408E

VGJ1408E: La impresora '%1' no se ha encontrado.\nEstán disponibles estas impresoras:\n%2

### Descripción

El usuario ha intentado imprimir en un dispositivo de impresora específico que no se ha encontrado en el sistema.

### Respuesta del usuario

Examine la configuración de impresoras del entorno. Asegúrese de que la impresora existe o imprima en otra impresora.

---

## Código de error de ejecución de Java EGL VGJ1409E

VGJ1409E: No existe ningún dispositivo de pantalla para los formularios.

### Descripción

No existe ningún dispositivo de pantalla para los formularios.

### Respuesta del usuario

Verifique que el programa de EGL se esté ejecutando en un entorno soportado de visualización y plataforma.

---

## Código de error de ejecución de Java EGL VGJ1410E

VGJ1410E: No existe ningún tamaño de dispositivo compatible para los formularios visualizados.

### Descripción

No existe ningún tamaño de dispositivo compatible para los formularios visualizados.

### Respuesta del usuario

Verifique que el programa de EGL se esté ejecutando en un entorno soportado de visualización y plataforma.

---

## Código de error de ejecución de Java EGL VGJ1411E

VGJ1411E: La clase de formulario de ayuda "%1" no existe.

### Descripción

Se ha intentado hacer referencia a una clase de formulario de ayuda que no existe.

### Respuesta del usuario

Verifique que la clase de formulario de ayuda esté definida y que la referencia se haga correctamente.

---

## Código de error de ejecución de Java EGL VGJ1412E

VGJ1412E: Atributo desconocido "%1".

### Descripción

El atributo especificado no se ha reconocido.

### Respuesta del usuario

Verifique que se esté utilizando el nombre de atributo correcto.

---

## Código de error de ejecución de Java EGL VGJ1414E

VGJ1414E: No se puede crear el formulario de ayuda '%1'

### Descripción

No se ha podido crear el formulario de ayuda que debe visualizarse.

### Respuesta del usuario

Asegúrese de que existen los grupos de formularios y formularios adecuados en la aplicación y de que se hayan generado correctamente.

---

## Código de error de ejecución de Java EGL VGJ1415E

VGJ1415E: ERROR INTERNO: %1

### Descripción

Se ha producido un error interno.

### Respuesta del usuario

Póngase en contacto con el soporte de técnico de IBM.

---

## Código de error de ejecución de Java EGL VGJ1416E

VGJ1416E: No hay impresoras disponibles.

### Descripción

El usuario ha intentado imprimir, pero el sistema no dispone de dispositivos de impresora utilizables.

### Respuesta del usuario

Imprima en un archivo o configure un dispositivo de impresora en el entorno.

---

## Código de error de ejecución de Java EGL VGJ1417E

VGJ1417E: No hay impresora por omisión.

### Descripción

El usuario ha intentado imprimir en la impresora por omisión, pero no se ha designado ninguna impresora por omisión en el sistema.

### Respuesta del usuario

Imprima en una impresora específica o defina la impresora por omisión en el entorno.

---

## Código de error de ejecución de Java EGL VGJ1419E

VGJ1419E: El valor formateado '%1' para '%2' es más largo que la longitud máxima (%3) permitida.

### Descripción

Existe un valor formateado (fecha, hora, moneda) que es demasiado largo para el campo.

### Respuesta del usuario

Aumente el tamaño del campo o formatee el valor con una longitud menor.

---

## Código de error de ejecución de Java EGL VGJ9900E

VGJ9900E: Se ha producido un error. El error es %1. No se puede cargar la descripción del error.

### Descripción

El programa no ha podido localizar o cargar el archivo de clase de mensaje por omisión y el archivo de clase de mensaje para su entorno local. Es posible que falte uno o ambos archivos de clase de mensaje o que estén dañados.

**Nota:** Durante la ejecución, este mensaje solamente visualizarse en inglés de EE.UU. debido al problema de carga de archivos de mensajes.

### Respuesta del usuario

Si ha extraído archivos de clase del archivo fda6.jar, verifique que las clases que tiene son del mismo release o nivel de mantenimiento que las clases del archivo más reciente. Si está utilizando clases más antiguas, sustitúyalas por la versión correcta. Además, puede volver a instalar fda6.jar desde EGL.

Si el problema persiste, haga lo siguiente:

1. Registre el número de mensaje y el texto del mensaje.

**Nota:** El mensaje de error incluye la siguiente información de importancia:

- Dónde se produjo el error
- El tipo de error interno

2. Registre la situación en la que aparece este mensaje.
3. Para obtener más instrucciones sobre cómo informar de posibles defectos al Centro de soporte de IBM, consulte la *Guía de instalación de EGL*.

---

## Código de error de ejecución de Java EGL VGJ9901E

**VGJ9901E:** Se ha producido un error. El error es %1. No se ha encontrado el texto de mensaje %1 en el archivo de clase de mensaje %2. Tampoco se ha encontrado el texto de mensaje para VGJ0002E.

### Descripción

El archivo de clase de mensaje no contiene el mensaje de entorno de ejecución para el ID de mensaje o para el ID de mensaje VGJ0002E. El archivo de clase de mensaje está corrompido o es de un release anterior de EGL.

**Nota:** Durante la ejecución, este mensaje solamente visualizarse en inglés de EE.UU. debido al problema de carga de archivos de mensajes.

### Respuesta del usuario

Si ha extraído archivos de clase del archivo fda6.jar, verifique que las clases que tiene son del mismo release o nivel de mantenimiento que las clases de ese archivo. Si está utilizando clases más antiguas, sustitúyalas por la versión correcta. Además, puede volver a instalar fda6.jar desde EGL.

Si el problema persiste, haga lo siguiente:

1. Registre el número de mensaje y el texto del mensaje.

**Nota:** El mensaje de error incluye la siguiente información de importancia:

- Dónde se produjo el error
  - El tipo de error interno
2. Registre la situación en la que aparece este mensaje.
  3. Para obtener más instrucciones sobre cómo informar de posibles defectos al Centro de soporte de IBM, consulte la *Guía de instalación de EGL*.

---

## Apéndice. Avisos

Derechos restringidos para los usuarios del gobierno de EE.UU.: El uso, la utilización, la duplicación o la divulgación están sujetos a las restricciones establecidas en el contrato GSA ADP Schedule Contract con IBM Corp.

Esta información se ha desarrollado para productos y servicios ofrecidos en los EE.UU. IBM puede no ofrecer los productos, servicios o características tratados en este documento en otros países. Póngase en contacto con el representante local de IBM que le informará sobre los productos y servicios disponibles actualmente en su área. Las referencias hechas a productos, programas o servicios de IBM no pretenden afirmar ni dar a entender que únicamente puedan utilizarse dichos productos, programas o servicios de IBM. Puede utilizarse en su lugar cualquier otro producto, programa o servicio funcionalmente equivalente que no vulnere ninguno de los derechos de propiedad intelectual de IBM. No obstante, es responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio que no sea de IBM.

IBM puede tener patentes o solicitudes de patente pendientes de aprobación que cubran temas descritos en este documento. La posesión de este documento no le otorga ninguna licencia sobre dichas patentes. Puede enviar consultas sobre las licencias, por escrito, a:

IBM Director of  
Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
Estados Unidos

Para consultas sobre licencias relativas a la información de doble byte (DBCS), póngase en contacto con el departamento de propiedad intelectual de IBM en su país o envíe las consultas, por escrito, a:

IBM World Trade Asia  
Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106-0032, Japón

**El párrafo siguiente no se aplica al Reino Unido ni a ningún otro país en que dichas disposiciones entren en contradicción con las leyes locales:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTA PUBLICACIÓN "TAL CUAL" SIN GARANTÍA DE NINGÚN TIPO, NI EXPLÍCITA NI IMPLÍCITA, INCLUYENDO, PERO NO LIMITÁNDOSE, A LAS GARANTÍAS IMPLÍCITAS DE NO VULNERABILIDAD, COMERCIALIZACIÓN O ADECUACIÓN A UN PROPÓSITO DETERMINADO. Algunas legislaciones no contemplan la declaración de limitación de responsabilidad, ni implícita ni explícita, en determinadas transacciones, por lo que cabe la posibilidad de que esta declaración no se aplique en su caso.

Esta información puede contener imprecisiones técnicas o errores tipográficos. Periódicamente se efectúan cambios en la información incluida en este documento; estos cambios se incorporarán en nuevas ediciones de la publicación. IBM puede

efectuar mejoras y/o cambios en los productos y/o programas descritos en esta publicación en cualquier momento y sin previo aviso.

Cualquier referencia hecha en esta información a sitios Web no de IBM se proporciona únicamente para su comodidad y no debe considerarse en modo alguno como promoción de esos sitios Web. Los materiales de estos sitios Web no forman parte de los materiales de IBM para este producto y el uso que se haga de estos sitios Web es de la entera responsabilidad del usuario.

IBM puede utilizar o distribuir la información que usted le suministre del modo que IBM considere conveniente sin incurrir por ello en ninguna obligación para con usted.

Los licenciarios de este programa que deseen obtener información acerca del mismo con el fin de: (i) intercambiar la información entre programas creados independientemente y otros programas (incluyendo éste) y (ii) utilizar mutuamente la información que se ha intercambiado, deben ponerse en contacto con:

IBM Corporation  
Lab Director  
IBM Canada Ltd. Laboratory  
8200 Warden Avenue  
Markham, Ontario, Canadá L6G 1C7

Esta información puede estar disponible, sujeta a los términos y condiciones adecuados, incluyendo en algunos casos el pago de una tarifa.

El programa bajo licencia descrito en esta información y todo el material bajo licencia disponible para el mismo, lo proporciona IBM bajo los términos del Acuerdo de Cliente IBM, el Acuerdo de Licencia de Programa Internacional IBM o cualquier otro acuerdo equivalente entre ambas partes.

Los datos de rendimiento incluidos aquí se determinaron en un entorno controlado. Por lo tanto, los resultados que se obtengan en otros entornos operativos pueden variar significativamente. Puede que se hayan tomado algunas medidas en los sistemas a nivel de desarrollo y no existe ninguna garantía de que estas medidas serán las mismas en sistemas disponibles generalmente. Además, puede que se haya estimado alguna medida mediante la extrapolación. Los resultados reales pueden variar. Los usuarios de este documento deben verificar los datos aplicables a su entorno específico.

La información concerniente a productos no IBM se ha obtenido de los suministradores de esos productos, de sus anuncios publicados o de otras fuentes de información pública disponibles. IBM no ha comprobado los productos y no puede afirmar la exactitud en cuanto a rendimiento, compatibilidad u otras características relativas a productos no IBM. Las consultas acerca de las posibilidades de productos no IBM deben dirigirse a los suministradores de los mismos.

Todas las declaraciones respecto a las intenciones futuras de IBM están sujetas a cambios o a su retirada sin aviso, y solamente representan metas y objetivos.

Todos los precios de IBM mostrados son precios actuales de venta al por menor sugeridos por IBM y sujetos a modificaciones sin previo aviso. Los precios de los concesionarios pueden ser diferentes.

Esta información es solamente para planificación. La información aquí contenida está sujeta a cambios antes de que los productos descritos estén disponibles.

Esta información contiene ejemplos de datos e informes utilizados en operaciones comerciales diarias. Para ilustrarlos tan completamente como sea posible, los ejemplos pueden incluir nombres de individuos, compañías, marcas y productos. Todos estos nombres son ficticios y cualquier parecido con los nombres y direcciones utilizados por una empresa real es pura coincidencia.

#### LICENCIA DE COPYRIGHT:

Esta información contiene programas de aplicación de ejemplo en lenguaje fuente, que muestran técnicas de programación en varias plataformas operativas. Puede copiar, modificar y distribuir estos programas de ejemplo de cualquier forma sin pagar nada a IBM, bajo el propósito de desarrollo, uso, marketing o distribución de programas de aplicación de acuerdo con la interfaz de programación de la aplicación para la plataforma operativa para la cual se han escrito los programas de ejemplo. Estos ejemplos no se han probado bajo todas las condiciones posibles. IBM, por lo tanto, no puede garantizar ni implicar la fiabilidad, servicio o funcionalidad de estos programas. Puede copiar, modificar y distribuir estos programas de ejemplo de cualquier forma sin pagar nada a IBM bajo el propósito de desarrollo, uso, marketing o distribución de programas de aplicación de acuerdo con los interfaces de programación de aplicaciones de IBM.

Cada copia o parte de estos programas de ejemplo o trabajos derivados de ellos, deben incluir el aviso de copyright siguiente:

© (nombre de empresa) (año). Algunas partes de este código se derivan de programas de ejemplo de IBM Corp. © Copyright IBM Corp. 2000, 2004. Reservados todos los derechos.

Si está viendo esta información en copia software, las fotografías y las ilustraciones a color podrían no aparecer.

---

## Información de interfaces de programación

La información de interfaces de programación está destinada a ayudarle a crear software de aplicaciones mediante este programa.

Las interfaces de programación de uso general le permiten escribir software de aplicaciones que obtengan los servicios de las herramientas de este programa.

Sin embargo, esta información puede contener información de diagnóstico, modificación y ajuste. La información de diagnóstico, modificación y ajuste se proporciona como ayuda para depurar el software de las aplicaciones.

**Aviso:** no utilice esta información de diagnóstico, modificación y ajuste como interfaz de programación porque está sujeta a cambios.

---

## Marcas registradas y marcas de servicio

Los términos siguientes son marcas registradas de International Business Machines Corporation en Estados Unidos y/o en otros países:

- AIX
- CICS
- CICS/ESA

- ClearCase
- DB2
- IBM
- IMS
- Informix
- iSeries
- MQSeries
- MVS
- OS/400
- RACF
- Rational
- VisualAge
- WebSphere
- z/OS

Intel es una marca registrada de Intel Corporation en los Estados Unidos y/o en otros países.

Java y todas las marcas basadas en Java son marcas registradas de Sun Microsystems, Inc., en los Estados Unidos y/o en otros países.

Linux es una marca registrada de Linus Torvalds en los Estados Unidos y/o en otros países.

Microsoft, Windows, y Windows NT son marcas registradas de Microsoft Corporation en los Estados Unidos y/o en otros países.

UNIX es una marca registrada de The Open Group en los Estados Unidos y/o en otros países.

Otros nombres de empresas, productos o nombres de servicio pueden ser marcas registradas o marcas de servicio de otros.

# Índice

## A

- abs() 840
- Accesos directos de teclado 129
- acos() 841
- action
  - Propiedad a nivel de campo primitivo 690
- activateWindow() 763
- activateWindowByName() 764
- activeForm 763
- activeWindow 764
- addReportData() 862
- addReportParameter() 862
- align
  - Propiedad a nivel de campo primitivo 690
- ANY 37
- aplicaciones básicas
  - iniciar 339
- aplicaciones de texto
  - códigos de datos modificados 161
  - componentes formGroup 153
  - iniciar 340
  - segmentación 160
- Aplicaciones Web
  - Page Designer 192
  - soporte 187
- appendAll() 75
- appendElement() 76
- archivos
  - asociaciones con tipos de registros 737
  - construcción 13, 293
  - crear 128, 293
  - definición de servicio Web 13
  - entorno J2EE 357
  - fuentes 13, 128
  - mandato EGL 481
  - propiedades de enlace 364, 657
  - propiedades del programa 350
  - resultados 328
  - suprimir en el Explorador de proyectos 278
- archivos de construcción
  - añadir sentencias de importación 319, 320
  - crear 293
  - descripción 13
  - editar sentencias de importación 319
  - formato 380
- archivos de definición de servicio Web 13
- archivos de entorno J2EE
  - actualizar 356
  - descripción 357
- archivos de mandatos 481
- archivos de mandatos EGL 481
- archivos de propiedades de enlace
  - descripción 364
  - desplegar 364
  - detalles 657

- archivos de propiedades del programa 350
- archivos de resultados 328
- archivos ear, eliminar archivos jar duplicados jar 355
- archivos fuente
  - ayuda de contenido 129, 483
  - comentar 275
  - crear 128
  - descripción 13
  - editores
    - comentar código fuente 275
  - formato 491
  - localizar en el Explorador de proyectos 278
- archivos jar, tiempo de ejecución
  - eliminar duplicados de archivos ear 355
  - proporcionar acceso a 365
- arrayIndex 929
- asignaciones 374
- asin() 842
- atan() 842
- atan2() 842
- attachBlobToFile() 832
- attachBlobToTempFile() 832
- attachClobToFile() 833
- attachClobToTempFile() 833
- autorización de base de datos 466
- ayuda de contenido
  - descripción 483
  - utilizar 129

## B

- base de datos predeterminada, SQL 251
- beginDatabaseTransaction() 889
- biblioteca curses, UNIX 352
- biblioteca curses UNIX 352
- Biblioteca de informes
  - registro Report 212
  - registro ReportData 212
  - visión general 861
- bibliotecas del sistema
  - DateLib 791
- BLOB 49
- bloques de establecimiento de valor 67
- byPassValidation
  - Propiedad a nivel de campo primitivo 691
- bytes() 890

## C

- calculateChkDigitMod10() 890
- calculateChkDigitMod11() 891
- callCmd() 893
- callConversionTable 930
- campos
  - ConsoleField 441

- campos (*continuación*)
  - estructura 752
  - Menu 455
  - MenuItem 456
  - PresentationAttributes 458
  - propiedades 64
  - propiedades, página 685
  - propiedades, SQL 67
  - Window 461
- Campos
  - Prompt 460
- cancelArrayDelete() 764
- cancelArrayInsert() 764
- característica de recuperación, SQL 229, 252
- carpetas, crear 127
- ceiling() 843
- clearActiveForm() 765
- clearActiveWindow() 765
- clearFields() 765
- clearFieldsByName() 765
- clearForm() 766
- clearRequestAttr() 802
- clearScreen() 789
- clearSessionAttr() 802
- clearWindow() 766
- clearWindowByName() 766
- clip() 870
- closeActiveWindow() 767
- closeWindow() 767
- closeWindowByName() 767
- código de tiempo de ejecución de EGL para Java, instalar 351
- códigos de datos, SQL 744
- códigos de datos modificados 161
- Códigos de error de ejecución de Java EGL 959
- colas de mensajes
  - llamadas directas de MQSeries 270
  - Palabras clave EGL relacionadas con MQSeries 268
  - Propiedades de registros MQ 665
  - registros de opciones MQ 665
  - remoto 269
  - Soporte de MQSeries 265
- color
  - Propiedad a nivel de campo primitivo 692
- column
  - Propiedad a nivel de campo primitivo 693
- comentarios 438
- comentarios, código fuente 275
- commentLine 768
- commit() 893
- commitOnConverse 922
- compareNum() 844
- compareStr() 870
- Compatibilidad de asignación 369
- Compatibilidad de referencia 739

- componente arrayDictionary
  - descripción 86
- componente de programa
  - propiedades 733
- componentes
  - abrir 277
  - buscar 275
  - descripción 17
  - propiedades 64
  - referencias a 21
- componentes dataItem
  - crear 131
  - descripción 131
  - formato fuente EGL 472
- componentes dataTable
  - crear 145
  - descripción 146
  - formato fuente EGL 473
- componentes de asociaciones de recursos
  - añadir 309
  - descripción 304
  - editar 309
  - elementos de asociaciones 375
  - eliminar 311
- componentes de biblioteca
  - crear 141
  - declaraciones de uso 955
  - formato fuente EGL 649
  - salida generada 649
- componentes de biblioteca, tipo basicLibrary
  - descripción 142
- componentes de biblioteca, tipo nativeLibrary
  - descripción 143
- componentes de construcción
  - asociaciones de recursos 304
  - descriptor de construcción 116, 293
  - opciones de enlace 311
- componentes de datos
  - dataItem 131, 472
  - dataTable 146, 473
  - registro básico 379
  - registro indexado 535
  - registro MQ 662
  - registro relativo 740
  - registro serie 743
  - Registro SQL 748
- componentes de descriptor de construcción
  - añadir 298
  - descripción 293
  - editar opciones generales 299
  - editar propiedades de tiempo de ejecución Java 302
  - eliminar 304
  - establecer el valor predeterminado 116
  - opciones, lista alfabética 382
  - opciones Java 300
- componentes de formulario
  - crear un formulario de impresión 155
  - crear un formulario de texto 158
  - crear un formulario en el editor de formularios de EGL 166
  - descripción 154
- componentes de formulario (*continuación*)
  - editar 164
  - filtrar 166, 176
  - formato fuente EGL 511
  - plantillas 171
  - print 156
  - propiedades de formato 66
  - propiedades de presentación de campos 66
  - propiedades de validación 67
  - text 158
- componentes de función 140, 527
  - crear 140
  - parámetros 522
  - variables 520
- componentes de interfaz de usuario (UI)
  - editar 164
  - formGroup 153, 508
  - formulario 154, 511
  - propiedades de campo de página 685
- componentes de opciones de enlace
  - añadir 313
  - descripción 311
  - editar elementos asynchLink 315
  - editar elementos callLink 314
  - editar elementos relacionados con transferencia 317
  - eliminar 318
- componentes de programa
  - básico 729
  - crear 138
  - datos no de parámetro 723
  - declaraciones de uso 955
  - descripción 139
  - formato fuente EGL 728
  - formularios de entrada 736
  - generación de envoltura Java 676
  - generación de programas Java 328
  - generación Java 675
  - parámetros 726
  - registros de entrada 736
  - textUI 731
- componentes de programa básico 729
- componentes de programa textUI 731
- componentes de registro
  - básico 379
  - crear 132
  - descripción 132
  - indexado 535
  - MQ 662
  - Page Designer 200
  - propiedades, longitud variable 737
  - relativo 740
  - serie 743
  - SQL 229, 253, 254, 748
- componentes de registro básico 379
- componentes de registro fijo
  - descripción 133
- componentes de registro indexado 535
- componentes de registro MQ
  - formato fuente EGL 662
  - propiedades 665
  - registros de opciones 665
- componentes de registro relativo 740
- componentes de registro serie 743
- componentes de registro SQL 748
- componentes descriptores de construcción
  - descriptores de construcción maestro 297
- componentes formGroup
  - crear 153
  - declaraciones de uso 957
  - descripción 153
  - editar 164
  - formato fuente EGL 508
  - Propiedad pfKeyEquate 686
- componentes lógicos
  - biblioteca 649
  - biblioteca, tipo basicLibrary 142
  - biblioteca, tipo nativeLibrary 143
  - función 140, 527
  - pageHandler 679
  - PageHandler 194
  - programa básico 729
  - programa textUI 731
- componentes pageHandler
  - componentes de selección múltiple 206
  - crear 191
  - declaraciones de uso 957
  - enlazar componentes de entrada 203
  - enlazar componentes de mandato 201
  - enlazar componentes de recuadro de selección 203
  - enlazar componentes de salida 203
  - enlazar componentes de selección única 204
  - formato fuente EGL 679
- componentes PageHandler
  - descripción 194
- concatenate() 872
- concatenateWithSeparator() 872
- conditionAsInt() 894
- conexiones JDBC
  - estándar 263
  - J2EE 362
- conexiones JDBC de J2EE 362
- conexiones JDBC estándar 263
- configuración de llamada de CICSJ2C 358
- configuración del despliegue, J2EE
  - conexiones JDBC 362
  - ConnectionFactory, CICSJ2C 358
  - entorno de tiempo de ejecución 354
  - escuchas de TCP/IP 353, 359
  - valores de descriptor 355, 357
- configuración del despliegue de J2EE
  - conexiones JDBC 362
  - ConnectionFactory, CICSJ2C 358
  - entorno de tiempo de ejecución 354
  - escuchas de TCP/IP 353, 359
  - valores de descriptor 355, 357
- configuraciones de lanzamiento
  - Escucha 287
  - explícito 286
  - implícito 285
- connect() 895
- ConnectionFactory, CICSJ2C 358
- connectionService() 916
- ConsoleField
  - campos 441
  - propiedades 441

- ConsoleForm
  - propiedades de componente 454
- ConsoleLib
  - activateWindow() 763
  - activateWindowByName() 764
  - activeForm 763
  - activeWindow 764
  - cancelArrayDelete() 764
  - cancelArrayInsert() 764
  - clearActiveForm() 765
  - clearActiveWindow 765
  - clearFields() 765
  - clearFieldsByName() 765
  - clearForm() 766
  - clearWindow() 766
  - clearWindowByName() 766
  - closeActiveWindow() 767
  - closeWindow() 767
  - closeWindowByName() 767
  - commentLinnet 768
  - currentArrayCount() 768
  - currentArrayDataLine() 768
  - currentArrayScreenLine() 768
  - currentDisplayAttrs 769
  - currentRowAttrs 769
  - cursorWrap 769
  - defaultDisplayAttributes 769
  - defaultInputAttributes 770
  - deferInterrupt 770
  - deferQuit 770
  - definedFieldOrder 770
  - displayAtLine() 771
  - displayAtPosition() 771
  - displayError() 771
  - displayFields() 772
  - displayFieldsByName() 772
  - displayForm() 772
  - displayFormByName() 773
  - displayLineMode() 773
  - displayMessage() 773
  - drawBox() 774
  - drawBoxWithColor() 774
  - errorLine 775
  - errorWindow 775
  - errorWindowVisible 775
  - formLine 775
  - getKey() 775
  - getKeyCode() 776
  - getKeyName() 776
  - gotoField() 776
  - gotoFieldByName() 777
  - gotoMenuItem() 777
  - gotoMenuItemByName() 777
  - hideAllMenuItems() 778
  - hideErrorWindow() 778
  - hideMenuItem() 778
  - hideMenuItemByName() 778
  - interruptRequested 779
  - isCurrentField() 779
  - isCurrentFieldByName() 779
  - isFieldModified() 779
  - isFieldModifiedByName() 780
  - key\_accept 780
  - key\_deleteLine 781
  - key\_help 781
  - key\_insertLine 781
  - key\_interrupt 781

- ConsoleLib (*continuación*)
  - key\_pageDown 781
  - key\_pageUp 782
  - key\_quit 782
  - lastKeyTyped() 782
  - menuLine 782
  - messageLine 782
  - messageResource 783
  - nextField() 783
  - openWindow() 783
  - openWindowByName() 783
  - openWindowWithForm() 784
  - openWindowWithFormByName() 784
  - previousField() 784
  - promptLine 785
  - promptLineMode() 785
  - quitRequested 785
  - screen 785
  - scrollDownLines() 785
  - scrollDownPage() 786
  - scrollUpLines() 786
  - scrollUpPage() 786
  - setArrayLine() 786
  - setCurrentArrayCount() 787
  - showAllMenuItems() 787
  - showHelp() 787
  - showMenuItem() 787
  - showMenuItemByName() 788
  - sqlInterrupt 788
- ConsoleUI 178
  - sentencia OpenUI 620
  - visión general 180
- constantes, declaraciones 53
- constantes, referencias a 58
- containsKey() 85
- ConverseLib
  - clearScreen() 789
  - displayMsgNum() 789
  - fieldInputLength() 790
  - pageEject() 790
  - validationFailed() 791
- ConverseVar
  - commitOnConverse 922
  - eventKey 923
  - printerAssociation 923
  - segmentedMode 925
  - validationMsgNum 926
- conversión
  - datos 467
  - texto de idioma bidireccional 470
- conversión de datos 467
- conversión de texto de idioma
  - bidireccional 470
- convert() 897
- copyStr() 873
- cos() 844
- cosh() 845
- Crear 178
- csouidpwd.properties 378
- currency
  - Propiedad a nivel de campo primitivo 694
- currencySymbol
  - Propiedad a nivel de campo primitivo 694
- currentArrayCount() 768
- currentArrayDataLine() 768

- currentArrayScreenLine() 768
- currentDate() 793
- currentDisplayAttrs 769
- currentFormattedGregorianDate 941
- currentFormattedJulianDate 942
- currentFormattedTime 943
- currentGregorianDate 944
- currentJulianDate 944
- currentRowAttrs 769
- currentShortGregorianDate 945
- currentShortJulianDate 945
- currentTime() 793
- currentTimeStamp() 794
- cursores, SQL 229
- cursorWrap 769

## CH

- CHAR 38
- characterAsInt() 870

## D

- DATE 41
- dateFormat
  - Propiedad a nivel de campo primitivo 695
- dateOf() 794
- DateTimeLib 791
  - currentDate() 793
  - currentTime() 793
  - currentTimeStamp() 794
  - dateOf() 794
  - dateValue() 795
  - dateValueFromGregorian() 795
  - dateValueFromJulian() 795
  - dayOf() 796
  - extend() 796
  - intervalValue() 797
  - intervalValueWithPattern() 797
  - mdy() 798
  - monthOf() 798
  - timeOf() 799
  - timestampFrom() 799
  - timestampValue() 799
  - timestampValueWithPattern() 800
  - timeValue() 801
  - weekdayOf() 801
  - yearOf() 801
- dateValue() 795
- dateValueFromGregorian() 795
- dateValueFromJulian() 795
- dayOf() 796
- DBCHAR 38
- DECIMAL 50
- declaraciones de uso 954
- declarar
  - constantes 53
  - variables 53
- defaultDateFormat 874
- defaultDisplayAttributes 769
- defaultInputAttributes 770
- defaultMoneyFormat 875
- defaultNumericFormat 875
- defaultTimeFormat 875
- defaultTimestampFormat 876

- deferInterrupt 770
  - deferQuit 770
  - defineDatabaseAlias() 899
  - definedFieldOrder 770
  - definiciones de tipo 27
  - depurador, EGL
    - acceso a base de datos SQL 279
    - crear una configuración de lanzamiento 286
    - crear una configuración de lanzamiento de escucha 287
    - descriptores de construcción 279
    - establecer preferencias 114
    - iniciar un programa 285
    - iniciar un servidor 288
    - iniciar una sesión Web 289
    - invocación desde el código generado 279
    - mandatos 279
    - preferencia de tipo de sistema 279
    - preparar un servidor 288
    - recomendaciones 279
    - recorrer paso a paso un programa 291
    - sentencias de llamada 279
    - utilizar puntos de interrupción 290
    - ver variables 292
    - visión general 279
  - Depurador EGL
    - acceso a base de datos SQL 279
    - crear una configuración de lanzamiento 286
    - crear una configuración de lanzamiento de escucha 287
    - descriptores de construcción 279
    - establecer preferencias 114
    - iniciar un programa 285
    - iniciar un servidor 288
    - iniciar una sesión Web 289
    - invocación desde el código generado 279
    - mandatos 279
    - opciones de codificación de caracteres 115
    - preferencia de tipo de sistema 279
    - preparar un servidor 288
    - puntos de interrupción 290
    - recomendaciones 279
    - recorrer paso a paso un programa 291
    - sentencias de llamada 279
    - ver variables 292
    - visión general 279
  - descriptores de construcción maestro
    - visión general 297
  - descriptores de construcción maestros
    - eglmaster.properties 490
    - plugin.xml 506
  - descriptores de despliegue
    - actualizar 357
    - valores de establecimiento 355
  - despliegue, aplicaciones Java fuera de J2EE 350
  - devolver valores a EGL 437
  - diagramas de sintaxis 755
  - diccionario
    - descripción 82
  - diccionario (*continuación*)
    - funciones
      - containsKey() 85
      - getKeys() 85
      - getValues() 85
      - insertAll() 86
      - removeAll() 86
      - removeElement() 86
      - size() 86
    - propiedades 84
    - directorios, generar en 335
    - disconnect() 900
    - disconnectAll() 901
    - displayAtLine() 771
    - displayAtPosition() 771
    - displayError() 771
    - displayFields() 772
    - displayFieldsByName() 772
    - displayForm() 772
    - displayFormByName() 773
    - displayLineMode() 773
    - displayMessage() 773
    - displayMsgNum() 789
    - displayName
      - Propiedad a nivel de campo primitivo 697
    - displayUse
      - Propiedad a nivel de campo primitivo 698
  - documento de diseño de informes XML
    - añadir a un paquete 219
    - tipos de datos en 216
    - visión general 209
  - Documento de diseño para informes
    - añadir a un paquete 219
    - tipos de datos en 216
    - visión general 209
  - drawBox() 774
  - drawBoxWithColor() 774
- ## E
- editor de formularios de EGL
    - opciones de visualización 174
    - preferencias 175
    - visión general 164
  - Editor EGL
    - ayuda de contenido 483
    - preferencias 117
    - visión general 483
  - editores
    - abrir un componente 277
    - ayuda de contenido 129, 483
    - EGL 483
    - localizar archivos fuente 278
    - preferencias, EGL 117
  - EGL, Kit de desarrollo de software (SDK de EGL) 333
  - EGLCMD 331, 332, 478
  - eglmaster.properties 490
  - eglpsh 477
  - EGLSDK 488
  - elementos asynchLink
    - descripción 377
    - package 379
    - recordName 379
  - elementos callLink
    - alias 409
    - conversionTable 409
    - ctgKeyStore 410
    - ctgKeyStorePassword 410
    - ctgLocation 410
    - ctgPort 411
    - descripción 407
    - JavaWrapper 411
    - library 412
    - linkType 412
    - location 413
    - luwControl 414
    - package 415
    - parmForm 416
    - pgmName 417
    - providerURL 417
    - refreshScreen 418
    - remoteBind 418
    - remoteComType 419
    - remotePgmType 421
    - serverID 422
    - type 423
  - elementos de asociaciones 375
  - elementos transferToProgram
    - alias 953
  - elementos transferToTransaction
    - alias 953
    - descripción 952
    - externallyDefined 953
  - enlaces 192
  - entorno de tiempo de ejecución, configuración de J2EE 354
  - entorno de trabajo, generación en 329, 330
  - envolturas Java
    - clases 551
    - descripción 301
    - generar 300
    - nombres de alias 670
    - salida de generación 676
    - utilizar 9
  - errorCode 931
  - errorLine 775
  - errorLog() 901
  - errorWindow 775
  - errorWindowVisible 775
  - escuchas de TCP/IP 353, 359
  - Especificadores de fecha, hora e indicación de la hora 45
  - estilos de fuente, preferencias 117
  - estructuras 26
  - eventKey 923
  - excepciones
    - Bloques try 94
    - manejo de 94
    - sistema EGL 94, 492
    - Valores de error de E/S 536
  - exp() 845
  - exportReport() 863
  - expresiones
    - descripción 495
    - fecha y hora 495
    - lógicas 88, 497
    - numéricas 88, 504
    - serie 88
    - text 505

expresiones de fecha y hora 495  
expresiones de texto 505  
expresiones lógicas 497  
expresiones numéricas 504  
extend() 796

## F

fieldInputLength() 790  
fieldLen  
    Propiedad a nivel de campo primitivo 698  
fill  
    Propiedad a nivel de campo primitivo 699  
fillCharacter  
    Propiedad a nivel de campo primitivo 699  
fillReport() 863  
findStr() 876  
floatingAssign() 845  
floatingDifference() 846  
floatingMod() 847  
floatingProduct() 847  
floatingQuotient() 848  
floatingSum() 848  
floor() 849  
formatDate() 877  
formatNumber() 878  
formato de archivo de construcción EGL 380  
Formato fuente EGL 491  
formatTime() 879  
formatTimeStamp() 880  
formConversionTable 932  
formLine 775  
formularios de entrada 736  
formularios de impresión 156  
formularios de texto 158  
fragmentos de código  
    autoRedirect 150  
    databaseUpdate 152  
    getClickedRowValue 151  
    insertar 149  
    setCursorFocus 150  
freeBlob() 833  
freeClob() 834  
frexp() 849  
función C  
    con EGL 424  
    invocar 424, 428, 432  
    pila de argumentos 434, 437  
funciones, acceso Java 806  
funciones BIGINT 427  
funciones C  
    DATE 429  
    DATETIME 430  
    DECIMAL 431  
    INTERVAL 430  
funciones DATE 429  
funciones DATETIME 430  
Funciones de acceso Java 806  
funciones DECIMAL 431  
funciones INTERVAL 430

## G

generación  
    archivos de mandatos EGL 331, 332  
    asistente 329  
    componentes de biblioteca 649  
    destino de directorio 335  
    EGLCMD 331, 332, 478  
    eglpsh 477  
    EGLSDK 333, 488  
    entorno de trabajo 330  
    envolturas Java 300  
    establecer  
        EGL\_GENERATORS\_PLUGINDIR 339  
    interfaz por lotes 331, 332, 333  
    opciones Java 300  
    proyectos EJB, código de despliegue 338  
    salida de envoltura Java 676  
    salida Java 328, 675  
    SDK EGL 333  
    tipos de salida 529, 530  
    visión general 323  
    vista Resultados 532  
generación de código, tipos 9  
getBlobLen() 834  
getClobLen() 834  
getCmdLineArg() 901  
getCmdLineArgCount() 902  
getField() 813  
getFieldValue() 865  
getKey() 775  
getKeyCode() 776  
getKeyName() 776  
getKeys() 85  
getMaxSize() 76  
getMessage() 902  
getNextToken() 881  
getProperty() 903  
getReportData() 865  
getReportParameter() 865  
getReportVariableValue() 866  
getRequestAttr() 803  
getSessionAttr() 803  
getSize() 76  
getStrFromClob() 835  
getSubStrFromClob() 835  
getVAGSysType() 919  
getValues() 85  
gotoField() 776  
gotoFieldByName() 777  
gotoMenuItem() 777  
gotoMenuItemByName() 777

## H

hacer referencia  
    componentes 21  
    constantes 58  
    variables 58  
handleHardIOErrors 946  
handleOverflow 946  
handleSysLibraryErrors 947  
help  
    Propiedad a nivel de campo primitivo 699  
HEX 39

hideAllMenuItems() 778  
hideErrorWindow() 778  
hideMenuItem() 778  
hideMenuItemByName() 778  
highlight  
    Propiedad a nivel de campo primitivo 700

## I

import 33  
Informes  
    añadir un documento de diseño 219  
    biblioteca 861  
    código de ejemplo de origen de datos 217  
    código para invocar informes 225  
    crear 210  
    documento de diseño XML 209  
    ejemplos de código para el manejador de informes 221  
    ejemplos de código para funciones de controlador 217  
    escribir código de controlador de informe 225  
    exportar 228  
    formatos de archivo exportados 228  
    generar después de crear 227  
    manejador de informes 213  
    orígenes de datos 212  
    plantillas para 220  
    tipos de datos en documentos de diseño XML 216  
    visión general 209  
    visión general de la creación y la generación 210  
Informix  
    consideraciones especiales 252  
inicialización, datos 471  
inicialización de datos 471  
inputRequired  
    Propiedad a nivel de campo primitivo 700  
inputRequiredMsgKey  
    Propiedad a nivel de campo primitivo 700  
insertAll() 86  
insertElement() 76  
instalación, código de tiempo de ejecución de EGL para Java 351  
integerAsChar() 883  
intensity  
    Propiedad a nivel de campo primitivo 701  
interfaz por lotes para la generación 331, 332, 333  
interior de registro, SQL 747  
interruptRequested 779  
INTERVAL 42  
intervalValue() 797  
intervalValueWithPattern() 797  
invocaciones de función 518  
invocar  
    función C 428  
invoke() 815

- isBoolean
  - Propiedad a nivel de campo primitivo 701
- isCurrentField() 779
- isCurrentFieldByName() 779
- isDecimalDigit
  - Propiedad a nivel de campo primitivo 702
- isFieldModified() 779
- isFieldModifiedByName() 780
- isHexDigit
  - Propiedad a nivel de campo primitivo 702
- isNull() 818
- isNullable
  - Propiedad a nivel de campo primitivo 702
- isObjId() 819
- isReadOnly
  - Propiedad a nivel de campo primitivo 703

## J

- J2EE, archivos de entorno
  - actualizar 356
  - descripción 357
- J2EELib
  - clearRequestAttr() 802
  - clearSessionAttr() 802
  - getRequestAttr() 803
  - getSessionAttr() 803
  - setRequestAttr() 804
  - setSessionAttr() 804
- JavaLib
  - getField() 813
  - invoke() 815
  - isNull() 818
  - isObjId() 819
  - qualifiedTypeName() 820
  - remove() 821
  - removeAll() 822
  - setField() 823
  - store() 824
  - storeCopy() 826
  - storeField() 827
  - storeNew() 829
- JavaServer Faces 198
- JSP 192

## K

- key\_accept 780
- key\_deleteLine 781
- key\_help 781
- key\_insertLine 781
- key\_interrupt 781
- key\_pageDown 781
- key\_pageUp 782
- key\_quit 782

## L

- lastKeyTyped() 782
- Ldexp() 850
- like 656

- límites del sistema 494
- lineWrap
  - Propiedad a nivel de campo primitivo 704
- loadBlobFromFile() 835
- loadClobFromFile() 836
- loadTable() 904
- LobLib 831
  - attachBlobToFile() 832
  - attachBlobToTempFile() 832
  - attachClobToFile() 833
  - attachClobToTempFile() 833
  - freeBlob() 833
  - freeClob() 834
  - getBlobLen() 834
  - getClobLen() 834
  - getStrFromClob() 835
  - getSubStrFromClob() 835
  - loadBlobFromFile() 835
  - loadClobFromFile() 836
  - setClobFromString() 836
  - setClobFromStringAtPosition() 837
  - truncateBlob() 837
  - truncateClob() 837
  - updateBlobToFile() 838
  - updateClobToFile() 838
- log() 850
- log10() 851
- lowerCase
  - Propiedad a nivel de campo primitivo 705
- lowerCase() 883

## LL

- llamadas de programa 9

## M

- manejador de componentes
  - crear 221
- manejador de informes
  - crear 221
- Manejador de informes
  - crear 221
  - ejemplos de código 221
  - funciones 214
  - funciones que puede invocar 215
  - visión general 213
- marcas registradas 1079
- masked
  - Propiedad a nivel de campo primitivo 705
- matches 660
- MathLib
  - abs() 840
  - acos() 841
  - asin() 842
  - atan() 842
  - atan2() 842
  - ceiling() 843
  - compareNum() 844
  - cos() 844
  - cosh() 845
  - exp() 845
  - floatingAssign() 845

- MathLib (*continuación*)
  - floatingDifference() 846
  - floatingMod() 847
  - floatingProduct() 847
  - floatingQuotient() 848
  - floatingSum() 848
  - floor() 849
  - frexp() 849
  - Ldexp() 850
  - log() 850
  - log10() 851
  - maximum() 851
  - minimum() 852
  - modf() 852
  - pow() 853
  - precision() 853
  - round() 854
  - sin() 855
  - sinh() 855
  - sqrt() 856
  - stringAsDecimal() 856
  - stringAsFloat() 857
  - stringAsInt() 857
  - tan() 858
  - tanh() 858
- matrices 74
  - campos de estructura 78
  - funciones 75
    - appendAll() 75
    - appendElement() 76
    - getMaxSize() 76
    - getSize() 76
    - insertElement() 76
    - removeAll() 76
    - removeElement() 77
    - resize() 77
    - reSizeAll() 77
    - setMaxSize() 77
    - setMaxSizes() 77
  - matrices dinámicas 74
- matrices de campos de estructura 78
- matrices dinámicas 74
- Matrices dinámicas 74
- matrices estáticas 74
- matrices multidimensionales 74
- matrices unidimensionales 74
- maximum() 851
- maximumSize() 904
- maxLen
  - Propiedad a nivel de campo primitivo 705
- MBCHAR 39
- mdy() 798
- Menu
  - campos 455
- MenuItem
  - campos 456
- menuLine 782
- messageLine 782
- messageResource 783
- minimum() 852
- minimumInput
  - Propiedad a nivel de campo primitivo 706
- minimumInputMsgKey
  - Propiedad a nivel de campo primitivo 706

modf() 852  
modified  
    Propiedad a nivel de campo primitivo 706  
monthOf() 798  
mqConditionCode 948  
MQSeries  
    llamadas directas 270  
    palabras clave relacionadas con EGL 268  
    Propiedades de registros MQ 665  
    registros de opciones MQ 665  
    soporte 265

## N

needsSOSI  
    Propiedad a nivel de campo primitivo 707  
newWindow  
    Propiedad a nivel de campo primitivo 707  
nextField() 783  
nombre JNDI, establecer para proyectos EJB 358  
nombres  
    alias 667, 669, 670  
    convenios 672  
nombres de alias  
    envolturas Java 670  
    Java 669  
    visión general 667  
nombres de alias Java 669  
Novedades de EGL 1  
Novedades de EGL 6.0 4  
Novedades del iFix de EGL 6.0 3  
nulas 229  
NUM 51  
NUMC 52  
numElementsItem  
    Propiedad a nivel de campo primitivo 708  
numericSeparator  
    Propiedad a nivel de campo primitivo 709

## O

opción buildPlan del descriptor de construcción 385  
opción cicsj2cTimeout del descriptor de construcción 386  
opción commentLevel del descriptor de construcción 386  
opción currencySymbol del descriptor de construcción 387  
opción dbms del descriptor de construcción 387  
opción de menú construir proyecto 326  
opción de menú reconstruir proyecto 326  
opción de menú reconstruir todo 326  
opción decimalSymbol del descriptor de construcción 387  
opción destDirectory del descriptor de construcción 388

opción destHost del descriptor de construcción 388  
opción destPassword del descriptor de construcción 389  
opción destPort del descriptor de construcción 389  
opción destUserID del descriptor de construcción 390  
opción eliminateSystemDependentCode del descriptor de construcción 390  
opción enableJavaWrapperGen del descriptor de construcción 391  
opción genDataTables del descriptor de construcción 391  
opción genDirectory del descriptor de construcción 391  
opción genFormGroup del descriptor de construcción 392  
opción genHelpFormGroup del descriptor de construcción 392  
opción genProject del descriptor de construcción 393  
opción genProperties del descriptor de construcción 394  
opción J2EE del descriptor de construcción 396  
opción linkage del descriptor de construcción 397  
opción nextBuildDescriptor del descriptor de construcción 397  
opción prep del descriptor de construcción 397  
opción resourceAssociations del descriptor de construcción 398  
opción sessionBeanID del descriptor de construcción 398  
opción sqlDB del descriptor de construcción 400  
opción sqlIID del descriptor de construcción 401  
opción sqlJDBCdriverClass del descriptor de construcción 401  
opción sqlJNDIName del descriptor de construcción 402  
opción sqlPassword del descriptor de construcción 403  
opción sqlValidationConnectionURL del descriptor de construcción 403  
opción system del descriptor de construcción 404  
opción targetNLS del descriptor de construcción 405  
opción VAGCompatibility del descriptor de construcción 405  
opción validateSQLStatements del descriptor de construcción 406  
opciones de pantalla ConsoleUI usuarios de UNIX 184  
opciones para la generación Java 300  
openWindow() 783  
openWindowByName() 783  
openWindowWithForm() 784  
openWindowWithFormByName() 784  
operador in 532  
operador isa 539

operadores  
    in 532  
    isa 539  
    precedencia 673  
origenes de datos para informes 212  
outline  
    Propiedad a nivel de campo primitivo 709  
overflowIndicator 932

## P

PACF 52  
Page Designer  
    componentes de entrada 203  
    componentes de mandato 201  
    componentes de recuadro de selección 203  
    componentes de salida 203  
    componentes de selección múltiple 206  
    componentes de selección única 204  
    enlaces 192  
    registros 200  
    soporte 192  
    tipos primitivos 199  
    vista Edición rápida, código de manejador de páginas 202  
pageEject() 790  
palabras clave  
    new 183  
palabras del sistema  
    aplicación Web 802  
Palabras del sistema de aplicaciones Web 802  
palabras del sistema de archivo y base de datos  
    recordName.resourceAssociation 859  
palabras reservadas  
    EGL 486  
Palabras reservadas EGL 486  
paquetes  
    crear 128  
    descripción 13  
    recomendaciones para 13  
parámetros, función 522  
parámetros, programa 726  
pattern  
    Propiedad a nivel de campo primitivo 710  
persistent  
    Propiedad a nivel de campo primitivo 710  
personalización de mensajes para el tiempo de ejecución de Java EGL 661  
pila de argumentos  
    función C 434, 437  
planes de construcción  
    descripción 327  
    invocar después de la generación 334  
plantillas, preferencias 118  
plugin.xml 506  
posibilidades, habilitar 122  
pow() 853  
precision() 853  
preferencia de recuperación, SQL 121

- preferencias
  - conexiones de base de datos SQL 119
  - Depurador EGL 114
  - editor de formularios de EGL 175
  - Editor EGL 117
  - EGL 113
  - entradas de la paleta del editor de formularios de EGL 170
  - estilos de fuente 117
  - Migración de EGL a EGL 110
  - plantillas 118
  - recuperación de SQL 121
  - text 113
- preferencias de conexión a base de datos 119
- PresentationAttributes
  - campos 458
- previousField() 784
- printerAssociation 923
- proceso de desarrollo 8
- proceso del conjunto de resultados, SQL 229, 743
- Prompt
  - Campos 460
- promptLine 785
- promptLineMode() 785
- propiedad alias de los elemento relacionados con la transferencia 953
- propiedad alias del elemento callLink 409
- propiedad conversionTable del elemento callLink 409
- propiedad ctgKeyStore del elemento callLink 410
- propiedad ctgKeyStorePassword del elemento callLink 410
- propiedad ctgLocation del elemento callLink 410
- propiedad ctgPort del elemento callLink 411
- propiedad externallyDefined del elemento transferToTransaction 953
- propiedad JavaWrapper del elemento callLink 411
- propiedad library del elemento callLink 412
- propiedad linkType del elemento callLink 412
- propiedad location del elemento callLink 413
- propiedad luwControl del elemento callLink 414
- propiedad package del elemento asynchLink 379
- propiedad package del elemento callLink 415
- propiedad parmForm del elemento callLink 416
- Propiedad pfKeyEquate 686
- propiedad pgmName del elemento callLink 417
- propiedad providerURL del elemento callLink 417
- propiedad recordName del elemento asynchLink 379
- propiedad refreshScreen del elemento callLink 418
- propiedad remoteBind del elemento callLink 418
- propiedad remoteComType del elemento callLink 419
- propiedad remotePgmType del elemento callLink 421
- propiedad serverID del elemento callLink 422
- propiedad type del elemento callLink 423
- propiedades
  - campo de página 685
  - campo SQL 67
  - campos 64
  - componente de programa 733
  - componentes 64
  - ConsoleField 441
  - formato 66
  - presentación de campos 66
  - registro MQ 665
  - registros de longitud variable 737
  - tiempo de ejecución Java 347, 540
  - validación 67
- Propiedades, nivel de campo primitivo
  - action 690
  - align 690
  - byPassValidation 691
  - color 692
  - column 693
  - currency 694
  - currencySymbol 694
  - dateFormat 695
  - displayName 697
  - displayUse 698
  - fieldLen 698
  - fill 699
  - fillCharacter 699
  - help 699
  - highlight 700
  - inputRequired 700
  - inputRequiredMsgKey 700
  - intensity 701
  - isBoolean 701
  - isDecimalDigit 702
  - isHexDigit 702
  - isNullable 702
  - isReadOnly 703
  - lineWrap 704
  - lowerCase 705
  - masked 705
  - maxLen 705
  - minimumInput 706
  - minimumInputMsgKey 706
  - modified 706
  - needsSOSI 707
  - newWindow 707
  - numElementsItem 708
  - numericSeparator 709
  - outline 709
  - pattern 710
  - persistent 710
  - protect 711
  - selectFromListItem 711
  - selectType 712
  - sign 712
  - sqlDataCode 713
  - sqlVariableLen 714
- Propiedades, nivel de campo primitivo (continuación)
  - timeFormat 715
  - timestampFormat 716
  - typeChkMsgKey 716
  - upperCase 717
  - validationOrder 717
  - validatorDataTable 718
  - validatorDataTableMsgKey 719
  - validatorFunction 719
  - validatorFunctionMsgKey 720
  - validValues 720
  - validValuesMsgKey 722
  - value 722
  - zeroFormat 722
- Propiedades a nivel de campo primitivo 686
  - action 690
  - align 690
  - byPassValidation 691
  - color 692
  - column 693
  - currency 694
  - currencySymbol 694
  - dateFormat 695
  - displayName 697
  - displayUse 698
  - fieldLen 698
  - fill 699
  - fillCharacter 699
  - help 699
  - highlight 700
  - inputRequired 700
  - inputRequiredMsgKey 700
  - intensity 701
  - isBoolean 701
  - isDecimalDigit 702
  - isHexDigit 702
  - isNullable 702
  - isReadOnly 703
  - lineWrap 704
  - lowerCase 705
  - masked 705
  - maxLen 705
  - minimumInput 706
  - minimumInputMsgKey 706
  - modified 706
  - needsSOSI 707
  - newWindow 707
  - numElementsItem 708
  - numericSeparator 709
  - outline 709
  - pattern 710
  - persistent 710
  - protect 711
  - selectFromListItem 711
  - selectType 712
  - sign 712
  - sqlDataCode 713
  - sqlVariableLen 714
  - timeFormat 715
  - timestampFormat 716
  - typeChkMsgKey 716
  - upperCase 717
  - validationOrder 717
  - validatorDataTable 718
  - validatorDataTableMsgKey 719

- Propiedades a nivel de campo primitivo (continuación)
  - validatorFunction 719
  - validatorFunctionMsgKey 720
  - validValues 720
  - validValuesMsgKey 722
  - value 722
  - zeroFormat 722
- propiedades de campo de página 685
- Propiedades de campo SQL 67
- propiedades de componente
  - ConsoleForm 454
- propiedades de EGL
  - visión general 64
- propiedades de formato 66
- propiedades de presentación de campos 66
- propiedades de tiempo de ejecución
  - Java 347, 540
- propiedades de validación 67
- protect
  - Propiedad a nivel de campo primitivo 711
- proyectos
  - crear 126
  - descripción 13
  - EJB, generación de código de despliegue 338
  - EJB, nombre JNDI 358
  - especificar opciones de base de datos 127
- proyectos EJB
  - establecer el nombre JNDI 358
  - generación de código de despliegue 338
- puntos de interrupción 290

## Q

- qualifiedTypeName() 820
- queryCurrentDatabase() 905
- quitRequested 785

## R

- recibir valores de EGL 434
- recomendaciones, desarrollo
  - asignación de componentes 13
  - descriptores de construcción 13
  - paquetes 13
- registros de entrada 736
- remove() 821
- removeAll() 76, 86, 822
- removeElement() 77, 86
- ReportLib
  - addReportData() 862
  - addReportParameter() 862
  - exportReport() 863
  - fillReport() 863
  - getFieldValue() 865
  - getReportData() 865
  - getReportParameter() 865
  - getReportVariableValue() 866
  - resetReportParameters() 866
  - setReportVariableValue() 866

- Requisitos de controlador JDBC en EGL 560
- resetReportParameters() 866
- resize() 77
- reSizeAll() 77
- resultSetID 743
- returnCode 933
- rollback() 905
- round() 854

## S

- salida
  - construir 327
  - generación de envoltura Java 676
  - generación Java 328, 675
  - opción de menú construir
    - proyecto 326
  - opción de menú reconstruir
    - proyecto 326
  - opción de menú reconstruir todo 326
  - tipos generados 529, 530
- screen 785
- scripts de construcción
  - descripción 342
  - opciones necesarias 406
- scrollDownLines() 785
- scrollDownPage() 786
- scrollUpLines() 786
- scrollUpPage() 786
- SDK EGL (kit de desarrollo de software de EGL) 333
- segmentación
  - aplicaciones de texto 160
- segmentedMode 925
- selectFromListItem
  - Propiedad a nivel de campo primitivo 711
- selectType
  - Propiedad a nivel de campo primitivo 712
- sentencia add 561
- Sentencia call 563
- sentencia case 566
- sentencia close 568
- sentencia continue 570
- Sentencia converse 570
- sentencia delete 571
- sentencia display 573
- sentencia execute 574
- sentencia exit 578
- sentencia for 580
- sentencia forEach 581
- sentencia forward 583
- sentencia freeSQ 584
- sentencia get 585
- sentencia get absolute 591
- sentencia get current 593
- sentencia get first 594
- sentencia get last 595
- sentencia get next 597
- sentencia get previous 602
- sentencia get relative 606
- sentencia goTo 608
- sentencia if, else 608
- sentencia move 610
- sentencia open 616

- Sentencia OpenUI 620
- sentencia prepare 630
- sentencia print 632
- sentencia replace 632
- sentencia return 635
- sentencia set 636
- sentencia show 646
- sentencia transfer 646
- sentencia try 648
- sentencia while 648
- sentencias
  - asignación 88
  - assignment 374
  - declaración de constante 88
  - declaración de variable 88
  - invocación de función 88, 518
  - nulas 88
  - palabra clave 88
  - SQL 229
- sentencias de palabra clave
  - add 561
  - call 563
  - case 566
  - close 568
  - converse 570
  - delete 571
  - display 573
  - execute 574
  - exit 578
  - for 580
  - forEach 581
  - forward 583
  - freeSQL 584
  - get 585
  - get absolute 591
  - get current 593
  - get first 594
  - get last 595
  - get next 597
  - get previous 602
  - get relative 606
  - goTo 608
  - if, else 608
  - lista alfabética 91
  - move 610
  - open 616
  - prepare 630
  - print 632
  - relacionadas con MQSeries 268
  - replace 632
  - return 635
  - set 636
  - show 646
  - transfer 646
  - try 648
  - while 648
- sentencias de palabras clave
  - continue 570
- sentencias SQL dinámicas 240
- sentencias SQL explícitas 260, 261, 262
- sentencias SQL implícitas 258, 259, 260, 261, 262
- servidores de construcción
  - descripción 343
  - iniciar en AIX, Linux o Windows 2000/NT/XP 343

- sesiones EJB
  - componentes 315
  - descripción 315
- sessionID 934
- setArrayLine() 786
- setBlankTerminator() 883
- setClobFromString() 836
- setClobFromStringAtPosition() 837
- setCurrentArrayCount() 787
- setCurrentDatabase() 906
- setError() 906
- setField() 823
- setLocale() 907
- setMaxSize() 77
- setMaxSizes() 77
- setNullTerminator() 884
- setRemoteUser() 908
- setReportVariableValue() 866
- setRequestAttr() 804
- setSessionAttr() 804
- setSubStr() 884
- showAllMenuItems() 787
- showHelp() 787
- showMenuItem() 787
- showMenuItemByName() 788
- sign
  - Propiedad a nivel de campo primitivo 712
- sin() 855
- sinh() 855
- size() 86, 909
- spaces() 885
- SQL
  - autorización de base de datos 466
  - base de datos predeterminada 251
  - característica de recuperación 229, 252
  - código de datos 744
  - componentes de registro 229
  - construir una sentencia PREPARE 259
  - crear componentes dataItem 253, 254
  - cursores 229
  - ejemplos 241
  - interior de registro 747
  - nulas 229
  - preferencias de conexión a base de datos 119
  - preferencias de recuperación 121
  - proceso del conjunto de resultados 229, 743
  - sentencias dinámicas 240
  - Sentencias EGL 229
  - sentencias explícitas 229, 260, 261, 262
  - sentencias implícitas 229, 258, 259, 260, 261, 262
  - sopORTE 229, 252
  - variables de sistema principal 744
- sqlca 935
- sqlcode 936
- sqlDataCode
  - Propiedad a nivel de campo primitivo 713
- sqlerrd 949
- sqlerrmc 950
- sqlInterrupt 788

- sqlIsolationLevel 950
- sqlState 936
- sqlVariableLen
  - Propiedad a nivel de campo primitivo 714
- sqlWarn 951
- sqrt() 856
- startCmd() 910
- startLog() 911
- startTransaction() 911
- store() 824
- storeCopy() 826
- storeField() 827
- storeNew() 829
- STRING 40
- stringAsDecimal() 856
- stringAsFloat() 857
- stringAsInt() 857
- strlen() 885
- StrLib
  - clip() 870
  - compareStr() 870
  - concatenate() 872
  - concatenateWithSeparator() 872
  - copyStr() 873
  - characterAsInt() 870
  - defaultDateFormat 874
  - defaultMoneyFormat 875
  - defaultNumericFormat 875
  - defaultTimeFormat 875
  - defaultTimestampFormat 876
  - findStr() 876
  - formatDate() 877
  - formatNumber() 878
  - formatTime() 879
  - formatTimeStamp() 880
  - getNextToken() 881
  - integerAsChar() 883
  - lowerCase() 883
  - setBlankTerminator() 883
  - setNullTerminator() 884
  - setSubStr() 884
  - spaces() 885
  - strlen() 885
  - textLen() 886
  - upperCase() 886
- Subseries 753
- SysLib
  - beginDatabaseTransaction() 889
  - bytes() 890
  - calculateChkDigitMod10() 890
  - calculateChkDigitMod11() 891
  - callCmd() 893
  - commit() 893
  - conditionAsInt() 894
  - connect() 895
  - convert() 897
  - defineDatabaseAlias() 899
  - disconnect() 900
  - disconnectAll() 901
  - errorLog() 901
  - getCmdLineArg() 901
  - getCmdLineArgCount() 902
  - getMessage() 902
  - getProperty() 903
  - loadTable() 904
  - maximumSize() 904

- SysLib (continuación)
  - queryCurrentDatabase() 905
  - rollback() 905
  - setCurrentDatabase() 906
  - setError() 906
  - setLocale() 907
  - setRemoteUser() 908
  - size() 909
  - startCmd() 910
  - startLog() 911
  - startTransaction() 911
  - unloadTable() 912
  - verifyChkDigitMod10() 913
  - verifyChkDigitMod11() 914
  - wait() 915
- systemType 937
- SysVar
  - arrayIndex 929
  - callConversionTable 930
  - errorCode 931
  - formConversionTable 932
  - overflowIndicator 932
  - returnCode 933
  - sessionId 934
  - sqlca 935
  - sqlcode 936
  - sqlState 936
  - systemType 937
  - terminalID 938
  - transactionID 939
  - transferName 939
  - userID 940

## T

- tan() 858
- tanh() 858
- terminalID 938
- textLen() 886
- texto, preferencias 113
- TIME 43
- timeFormat
  - Propiedad a nivel de campo primitivo 715
- timeOf() 799
- TIMESTAMP 44
- timeStampFormat
  - Propiedad a nivel de campo primitivo 716
- timeStampFrom() 799
- timeStampValue() 799
- timeStampValueWithPattern() 800
- timeValue() 801
- tipo primitivo CLOB 48
- tipos de datos C 428
- tipos de datos I4GL 428
- tipos de referencia 183
- tipos de registros
  - asociaciones con tipos de archivo 737
  - descripción 135
- tipos primitivos
  - ANY 37
  - BIN 50
  - BLOB 49
  - CLOB 48
  - CHAR 38
  - DATE 41

tipos primitivos (continuación)

- DBCHAR 38
- DECIMAL 50
- descripción 34
- FLOAT 51
- HEX 39
- INTERVAL 42
- MBCHAR 39
- MONEY 51
- NUM 51
- NUMC 52
- PACF 52
- Page Designer 199
- SMALLFLOAT 53
- STRING 40
- TIME 43
- TIMESTAMP 44
- UNICODE 40

tipos primitivos EGL 428

transactionID 939

transferencia de control entre programas 93

transferencias de programas 9

transferName 939

truncateBlob() 837

truncateClob() 837

typeChkMsgKey

- Propiedad a nivel de campo primitivo 716

typedef 27

## U

- UNICODE 40
- unidad lógica de trabajo 307
- unidades de ejecución 742
- unloadTable() 912
- updateBlobToFile() 838
- updateClobToFile() 838
- upperCase
  - Propiedad a nivel de campo primitivo 717
- upperCase() 886
- userID 940
- usuarios de UNIX
  - opciones de pantalla ConsoleUI 184

## V

- validationFailed() 791
- validationMsgNum 926
- validationOrder
  - Propiedad a nivel de campo primitivo 717
- validatorDataTable
  - Propiedad a nivel de campo primitivo 718
- validatorDataTableMsgKey
  - Propiedad a nivel de campo primitivo 719
- validatorFunction
  - Propiedad a nivel de campo primitivo 719
- validatorFunctionMsgKey
  - Propiedad a nivel de campo primitivo 720

- validValues
  - Propiedad a nivel de campo primitivo 720
- validValuesMsgKey
  - Propiedad a nivel de campo primitivo 722
- Valores de error de E/S 536
- value
  - Propiedad a nivel de campo primitivo 722
- variable de Interfaz de usuario de consola
  - errorLine 775
- variable
  - EGL\_GENERATORS\_PLUGINDIR 339
- variables, declaraciones 53
- variables, referencias a 58
- variables de sistema principal, SQL 744
- verifyChkDigitMod10() 913
- verifyChkDigitMod11() 914
- VGLib
  - connectionService() 916
  - getVAGSysType() 919
- VGVar
  - currentFormattedGregorianDate 941
  - currentFormattedJulianDate 942
  - currentFormattedTime 943
  - currentGregorianDate 944
  - currentJulianDate 944
  - currentShortGregorianDate 945
  - currentShortJulianDate 945
  - handleHardIOErrors 946
  - handleOverflow 946
  - handleSysLibraryErrors 947
  - mqConditionCode 948
  - sqlerrd 949
  - sqlerrmc 950
  - sqlIsolationLevel 950
  - sqlWarn 951
- vías de acceso de construcción, EGL
  - editar 320
  - visión general 477
- vías de acceso de construcción EGL
  - editar 320
  - visión general 477
- visión general de ConsoleUI 177
- visión general de EGL 1
- Visión general de Interfaz de usuario de consola 177
- vista Edición rápida
  - código de manejador de páginas 202
- vista Resultados, generación 532
- VisualAge Generator
  - compatibilidad de EGL 439
  - migración desde 12
- VSAM
  - nombres de sistema 264
  - requisitos previos de acceso 264
  - soporte 264

## W

- wait() 915
- weekdayOf() 801
- Window
  - campos 461

## Y

- yearOf() 801

## Z

- zeroFormat
  - Propiedad a nivel de campo primitivo 722







Número de Programa: 5724-J19

Impreso en España

SC11-3009-02

