

Rational Application Developer



EGL 参考指南

版本 6 发行版 001

Rational Application Developer



EGL 参考指南

版本 6 发行版 001

注意

在使用本资料及其支持的产品之前，请阅读第 989 页的『声明』中的信息。

第三版（2005 年 4 月）

此版本适用于 Rational Web Developer 和 Rational Application Developer V6R0M0 及所有后续发行版和修订版，除非新版本中另有指示。

© Copyright International Business Machines Corporation 1996, 2005. All rights reserved.

目录

概述	1	动态数组	68
EGL 简介	1	结构字段数组	71
EGL 6.0.0.1 中的新增内容	1	字典	75
EGL 6.0 iFix 中的新增内容	2	字典属性	76
EGL V6.0 中的新增内容	4	字典函数	78
开发过程	7	ArrayDictionary	79
运行时配置	8	EGL 语句	80
使用 Java 包装器	9	按字母顺序排列的关键字	83
有效调用	9	在程序之间转移控制权	85
有效转移	10	异常处理	86
有关 EGL 的其它信息的来源	11	try 块	86
		EGL 系统异常	87
EGL 语言概述	13	try 块的限制	88
EGL 项目、包和文件	13	与错误相关的系统变量	88
EGL 项目	13	I/O 语句	90
包	14	错误标识	90
EGL 文件	14		
建议	15	将 EGL 代码迁移至 EGL 6.0 iFix	93
部件	16	EGL 至 EGL 迁移	94
对部件的引用	20	EGL 至 EGL 迁移期间的属性更改	97
固定结构	23	设置 EGL 至 EGL 迁移首选项	102
Typedef	25		
Import	30	设置环境	105
背景知识	30	设置 EGL 首选项	105
Import 语句的格式	30	设置文本首选项	105
基本类型	31	为 EGL 调试器设置首选项	106
声明时的基本类型	33	设置缺省构建描述符	108
不同数字类型的相对效率	33	设置 EGL 编辑器的首选项	108
ANY	34	设置源代码样式首选项	108
字符类型	35	设置模板首选项	109
DateTime 类型	38	设置 SQL 数据库连接首选项	110
LOB 类型	44	设置 SQL 检索首选项	112
数字类型	46	启用 EGL 功能	113
声明 EGL 中的变量和常量	49		
动态和静态访问	50	开始开发代码	115
EGL 中的确定作用域规则和“this”	52	创建项目	115
引用 EGL 中的变量	53	创建 EGL 项目	115
动态访问的括号语法	55	创建 EGL Web 项目	116
引用固定结构的缩写语法	57	在创建项目时指定数据库选项	117
EGL 属性概述	59	创建 EGL 源文件夹	117
字段显示属性	60	创建 EGL 包	117
定义格式的属性	61	创建 EGL 源文件	118
SQL 项属性	61	将 EGL 模板与内容辅助配合使用	118
验证属性	62	EGL 的键盘快捷键	119
set-value 块	62		
基本情况的 set-value 块	62	开发基本 EGL 源代码	121
字段的字段的 set-value 块	64	创建 EGL dataItem 部件	121
“this”的用法	65	DataItem 部件	121
set-value 块、数组和数组元素	66	创建 EGL 记录部件	122
其它示例	66	记录部件	122
数组	68	固定记录部件	123

记录类型和属性	125
创建 EGL 程序部件	127
程序部件	128
创建 EGL 函数部件	129
函数部件	130
创建 EGL 库部件	130
类型为 basicLibrary 的库部件	131
类型为 nativeLibrary 的库部件	132
创建 EGL dataTable 部件	134
DataTable	134

将代码段插入到 EGL 和 JSP 文件中 137

将焦点设置为表单字段	138
针对会话变量测试浏览器	138
检索数据表中被单击的行的值	139
更新关系表中的行	139

使用文本和打印表单 141

创建 EGL formGroup 部件	141
FormGroup 部件	141
表单部件	142
创建 EGL 打印表单	143
创建 EGL 文本表单	145
EGL 表单编辑器概述	150
使用 EGL 表单编辑器编辑表单组	151
创建过滤器	152
在 EGL 表单编辑器中创建表单	152
创建常量字段	153
在打印或文本表单中创建变量字段	154
设置 EGL 表单编辑器选用板条目的首选项	156
EGL 表单编辑器中的表单模板	156
EGL 表单编辑器的显示选项	160
设置 EGL 表单编辑器的首选项	160
EGL 表单编辑器中的表单过滤器	161

创建控制台用户界面 163

控制台用户界面	163
使用 consoleUI 创建界面	164
ConsoleUI 部件和相关变量	165
Window (窗口)	166
Prompt (提示)	166
ConsoleField	166
ConsoleForm	167
ConsoleUI 中 new 的用法	168
UNIX 的 ConsoleUI 屏幕选项	169

创建 EGL Web 应用程序 171

Web 支持	171
创建单个表 EGL Web 应用程序	171
EGL 数据部件和页面向导	171
创建单个表 EGL Web 应用程序	172
在 EGL 数据部件和页面向导中定义 Web 页面	174
创建 EGL pageHandler 部件	175
EGL 的 Page Designer 支持	175
PageHandler	177
JavaServer Faces 控件和 EGL	180

创建 EGL 字段并将其与 Faces JSP 相关联	181
将 EGL 记录与 Faces JSP 相关联	182
将 JavaServer Faces 命令组件与 EGL PageHandler 绑定	183
对 PageHandler 代码使用“快速编辑”视图	184
将 JavaServer Faces 输入或输出组件与 EGL PageHandler 绑定	184
将 JavaServer Faces 复选框组件与 EGL PageHandler 绑定	185
将 JavaServer Faces 单选组件与 EGL PageHandler 绑定	186
将 JavaServer Faces 多选组件与 EGL PageHandler 绑定	187

创建 EGL 报告 189

EGL 报告概述	189
EGL 报告创建过程概述	190
数据源	191
库中的数据记录	192
EGL 报告处理程序	193
预定义报告处理程序函数	194
其它 EGL 报告处理程序函数	195
XML 设计文档中的数据类型	196
EGL 报告驱动程序函数的样本代码	197
将设计文档添加至包	198
使用报告模板	199
创建 EGL 报告处理程序	200
手工创建 EGL 报告处理程序	201
编写用于生成报告的代码	205
为报告生成文件并运行报告	206
导出报告	207

使用文件和数据库 209

SQL 支持	209
EGL 语句和 SQL	209
结果集处理	212
SQL 记录和它们的使用	214
声明时的数据库访问	218
动态 SQL	219
SQL 示例	219
缺省数据库	230
Informix 和 EGL	230
特定于 SQL 的任务	231
检索 SQL 表数据	231
从 SQL 记录部件创建数据项部件 (概述)	232
从关系数据库表创建 EGL 数据部件	233
查看 SQL 记录的 SQL SELECT 语句	236
验证 SQL 记录的 SQL SELECT 语句	236
构造 EGL prepare 语句	237
从隐式 SQL 语句构造显式 SQL 语句	237
重置显式 SQL 语句	239
从与 SQL 相关的 EGL 语句中除去 SQL 语句	239
解析引用以显示隐式 SQL 语句	239
了解如何建立标准 JDBC 连接	240
VSAM 支持	241
访问先决条件	241

系统名称	241
MQSeries 支持	242
连接	242
将消息包括在事务中	243
定制	243
与 MQSeries 相关的 EGL 关键字	244
直接 MQSeries 调用	246
维护 EGL 代码	251
对 EGL 源代码进行行注释	251
搜索部件	251
查看部件引用	252
在 .egl 文件中打开部件	252
在“项目资源管理器”中定位 EGL 源文件	253
在“项目资源管理器”中删除 EGL 文件	254
调试 EGL 代码	255
EGL 调试器	255
调试器命令	255
构建描述符的使用	257
SQL 数据库访问	258
call 语句	258
在调试时使用的系统类型	259
EGL 调试器端口	259
建议	259
调试 J2EE 之外的应用程序	260
在 EGL 调试器中启动非 J2EE 应用程序	260
在 EGL 调试器中创建启动配置	261
创建 EGL 侦听器启动配置	262
调试 J2EE 应用程序	262
为进行 EGL Web 调试而准备服务器	262
为进行 EGL Web 调试而启动服务器	263
启动 EGL Web 调试会话	263
在 EGL 调试器中使用断点	264
在 EGL 调试器中单步执行应用程序	265
在 EGL 调试器中查看变量	266
使用 EGL 构建部件	267
创建构建文件	267
设置一般构建选项	267
设置外部文件、打印机和队列关联	277
设置调用和传送选项	282
设置对其它 EGL 构建文件的引用	289
编辑 EGL 构建路径	290
生成、编译和运行 EGL 输出	293
生成	293
将 Java 代码生成到项目中	293
构建	295
构建 EGL 输出	296
构建规划	297
Java 程序、PageHandler 和库	297
结果文件	298
在工作台中生成	298
工作台中的生成	300
从工作台批处理接口生成	301

从工作台批处理接口生成	301
从 EGL 软件开发包 (SDK) 生成	302
从 EGL 软件开发包 (SDK) 生成	302
在生成之后调用构建规划	303
生成 Java; 其它主题	304
处理生成到目录中的 Java 代码	304
生成 EJB 项目的部署代码	307
设置变量 EGL_GENERATORS_PLUGINDIR	307
在本地机器上运行 EGL 生成的 Java 代码	308
在本地机器上启动基本或文本用户界面 Java 应用程序	308
在本地机器上启动 Web 应用程序	308
构建脚本	310
Java 构建脚本	310
构建服务器	311
在 AIX、Linux 或 Windows 2000/NT/XP 上启动构建服务器	311
部署 EGL 生成的 Java 输出	315
Java 运行时属性	315
在 J2EE 环境中	315
在非 J2EE Java 环境中	315
构建描述符和程序属性	316
有关其它信息	317
为 EGL 生成的代码设置非 J2EE 运行时环境	317
程序属性文件	317
在 J2EE 外部部署 Java 应用程序	318
安装用于 Java 的 EGL 运行时代码	318
将 JAR 文件包括在目标机器的 CLASSPATH 中	319
为 EGL 运行时设置 UNIX curses 库	319
为调用的非 J2EE 应用程序设置 TCP/IP 侦听器	320
为 EGL 生成的代码设置 J2EE 运行时环境	321
消除重复的 jar 文件	322
设置部署描述符值	322
更新 J2EE 环境文件	323
手工更新部署描述符	324
设置 EJB 项目的 JNDI 名称	325
为 CICSJ2C 调用设置 J2EE 服务器	325
为 J2EE 应用程序客户机模块中的被调用程序设置 TCP/IP 侦听器	326
设置 J2EE JDBC 连接	328
部署链接属性文件	329
提供对非 EGL jar 文件的访问	330
EGL 引用	335
EGL 中的赋值兼容性	335
在数字类型之间赋值	336
其它交叉类型赋值	336
字符类型的填充和截断	337
时间戳记之间的赋值	338
赋值给具有子结构的字段或从具有子结构的字段进行赋值	338
固定记录的赋值	339
赋值	340
关联元素	340
commit	341

conversionTable	341	EGL consoleUI 中的 ConsoleForm 属性	413
fileType	341	EGL consoleUI 中的 Menu 字段	414
fileName	341	EGL consoleUI 中的 MenuItem 字段	415
formFeedOnClose	341	EGL consoleUI 中的 PresentationAttributes 字段	417
replace	342	EGL consoleUI 中的 Prompt 字段	419
system	342	EGL consoleUI 中的 Window 字段	420
systemName	342	containerContextDependent	425
text	342	数据库权限和表名	425
asynchLink 元素	343	数据转换	426
远程调用的 csouidpwd.properties 文件	343	当调用程序是 Java 代码时的数据转换	427
asynchLink 元素中的 package	344	转换算法	428
asynchLink 元素中的 recordName	344	双向语言文本	429
EGL 源格式的基本记录部件	345	数据初始化	430
构建部件	346	EGL 源格式的 DataItem 部件	431
EGL 构建文件格式	346	EGL 源格式的 DataTable 部件	432
构建描述符选项	347	EGL 构建路径和 eglpath	435
构建脚本	369	EGLCMD	436
EGL 构建脚本中的必需选项	369	语法	436
callLink 元素	370	示例	438
如果 callLink 类型是 localCall (缺省值)	370	EGL 命令文件	439
如果 callLink 类型是 remoteCall	370	命令文件的示例	440
如果 callLink 类型是 ejbCall	371	EGL 编辑器	441
callLink 元素中的 alias	372	EGL 中的内容辅助	441
callLink 元素中的 conversionTable	372	EGL 中的枚举	441
callLink 元素中的 ctgKeyStore	373	EGL 保留字	444
callLink 元素中的 ctgKeyStorePassword	373	在 SQL 语句之外的保留字	444
callLink 元素中的 ctgLocation	373	EGLSDK	446
callLink 元素中的 ctgPort	374	语法	446
callLink 元素中的 JavaWrapper	374	示例	447
callLink 元素中的 linkType	375	eglmaster.properties 文件的格式	448
callLink 元素中的 library	375	EGL 源格式	448
callLink 元素中的 location	376	EGL 系统异常	450
callLink 元素中的 luwControl	377	EGL 系统限制	451
callLink 元素中的 package	378	表达式	452
callLink 元素中的 parmForm	378	日期时间表达式	453
callLink 元素中的 pgmName	379	逻辑表达式	454
callLink 元素中的 providerURL	379	数字表达式	461
callLink 元素中的 refreshScreen	380	文本表达式	462
callLink 元素中的 remoteBind	381	主构建描述符 plugin.xml 文件的格式	463
callLink 元素中的 remoteComType	381	EGL 源格式的 FormGroup 部件	464
callLink 元素中的 remotePgmType	383	屏幕浮动区域的属性	466
callLink 元素中的 serverID	384	打印浮动区域的属性	466
callLink 元素中的 type	385	EGL 源格式的表单部件	467
将 C 函数与 EGL 配合使用	386	文本表单属性	468
C 的 BIGINT 函数	389	打印表单属性	469
C 数据类型和 EGL 基本类型	390	表单字段	469
C 的 DATE 函数	391	文本表单字段属性	470
C 的 DATETIME 和 INTERVAL 函数	391	函数调用	473
C 的 DECIMAL 函数	392	函数变量	475
从 EGL 程序调用 C 函数	393	函数参数	477
C 的堆栈函数	395	inOut 及相关修饰符的含义	479
返回 C 的函数	397	EGL 源格式的函数部件	481
注释	399	生成的输出	483
与 VisualAge Generator 的兼容性	400	生成的输出 (参考)	484
ConsoleUI	401	“生成结果”视图	485
ConsoleField 属性和字段	401	in 运算符	486

有关一维数组的示例	487	while	591
有关多维数组的示例	487	库 (生成的输出)	591
EGL 源格式的带索引记录部件	488	EGL 源格式的库部件	592
I/O 错误值	490	like 运算符	597
duplicate	490	链接属性文件 (详细信息)	598
endOfFile	491	如果在运行时标识链接属性文件	598
format	491	链接属性文件的格式	599
noRecordFound	492	matches 运算符	601
unique	492	EGL Java 运行时的消息定制	602
isa 运算符	493	EGL 源格式的 MQ 记录部件	603
Java 运行时属性 (详细信息)	493	MQ 记录属性	606
Java 包装器类	502	队列名	606
关于如何使用包装器类的概述	503	将消息包括在事务中	606
程序包装器类	504	打开输入队列以供独占使用	606
参数包装器类集	505	MQ 记录的选项记录	606
具有子结构的项数组包装器类集	506	为名称取别名	608
动态数组包装器类	507	对 JSP 文件和生成的 Java bean 中的 EGL 标识	
Java 包装器类的命名约定	508	的更改	609
数据类型交叉引用	509	如何为 Java 名称取别名	610
EGL 中的 JDBC 驱动程序需求	510	如何为 Java 包装器名取别名	610
关键字	511	命名约定	612
add	511	运算符和优先顺序	613
call	513	Java 程序生成输出	615
case	515	Java 包装器生成输出	616
close	517	示例	617
continue	519	EGL 源格式的 PageHandler 部件	619
converse	519	PageHandler 部件属性	622
delete	520	PageHandler 字段属性	624
display	522	pfKeyEquate	625
execute	522	基本字段级别属性	625
exit	526	action	628
for	528	align	629
forEach	530	byPassValidation	629
forward	531	color	630
freeSQL	532	column	631
get	533	currency	632
get absolute	538	currencySymbol	632
get current	540	dateFormat	633
get first	541	displayName	635
get last	543	displayUse	635
get next	544	fieldLen	636
get previous	549	fill	637
get relative	552	fillCharacter	637
goTo	554	help	637
if, else	555	highlight	638
move	556	inputRequired	638
open	561	inputRequiredMsgKey	638
openUI	565	intensity	639
prepare	574	isBoolean	639
print	576	isDecimalDigit	639
replace	576	isHexDigit	640
return	579	isNullable	640
set	579	isReadOnly	641
show	588	lineWrap	642
transfer	589	lowerCase	642
try	590	masked	642

maxLen	643
minimumInput	643
minimumInputMsgKey	644
modified	644
needsSOSI	644
newWindow	645
numElementsItem	646
numericSeparator	646
outline	646
pattern	647
persistent	647
protect	648
selectFromListItem	649
selectType	649
sign	650
sqlDataCode	650
sqlVariableLen	651
timeFormat	652
timeStampFormat	653
typeChkMsgKey	654
upperCase	654
validationOrder	655
validatorDataTable	655
validatorDataTableMsgKey	656
validatorFunction	656
validatorFunctionMsgKey	657
validValues	658
validValuesMsgKey	659
value	659
zeroFormat	660
参数以外的程序数据	660
程序参数	663
EGL 源格式的程序部件	664
EGL 源格式的基本程序	666
EGL 源格式的文本用户界面程序	668
程序部件属性	670
输入表单	672
输入记录	672
记录和文件类型交叉引用	673
支持变长记录的属性	673
具有 lengthItem 属性的变长记录	673
具有 numElementsItem 属性的变长记录	674
同时具有 lengthItem 和 numElementsItem 属性的变长记录	675
在进行调用或转移时传递的变长记录	675
EGL 中的引用兼容性	675
EGL 源格式的相关记录部件	676
运行单元	678
resultSetID	678
EGL 源格式的串行记录部件	679
SQL 数据代码和 EGL 主变量	680
变长列和定长列	680
SQL 数据类型与 EGL 基本类型的兼容性	681
VARCHAR、VARGRAPHIC 和相关 LONG 数据类型	682
DATE、TIME 和 TIMESTAMP	682

SQL 记录内部结构	682
EGL 源格式的 SQL 记录部件	683
EGL 源格式中的结构字段	686
子串	688
EGL 函数的语法图	689
EGL 语句和命令的语法图	690
系统库	691
EGL 库 ConsoleLib	691
EGL 库 ConverseLib	720
EGL 库 DateTimeLib	723
EGL 库 J2EELib	733
EGL 库 JavaLib	737
EGL 库 LobLib	760
EGL 库 MathLib	767
recordName.resourceAssociation	786
EGL 库 ReportLib	788
EGL 库 StrLib	794
EGL 库 SysLib	813
EGL 库 VGLib	839
EGL 库外部的系统变量	844
ConverseVar	844
SysVar	849
VGVar	862
transferToTransaction 元素	873
与转移相关的链接元素中的 alias	874
transferToTransaction 元素中的 externallyDefined	874
使用声明	875
背景知识	875
在程序部件或库部件中	875
在 formGroup 部件中	877
在 pageHandler 部件中	878
EGL Java 运行时错误代码	879
EGL Java 运行时错误代码 CSO7000E	880
EGL Java 运行时错误代码 CSO7015E	881
EGL Java 运行时错误代码 CSO7016E	881
EGL Java 运行时错误代码 CSO7020E	881
EGL Java 运行时错误代码 CSO7021E	881
EGL Java 运行时错误代码 CSO7022E	882
EGL Java 运行时错误代码 CSO7023E	882
EGL Java 运行时错误代码 CSO7024E	882
EGL Java 运行时错误代码 CSO7026E	882
EGL Java 运行时错误代码 CSO7045E	882
EGL Java 运行时错误代码 CSO7050E	883
EGL Java 运行时错误代码 CSO7060E	883
EGL Java 运行时错误代码 CSO7080E	883
EGL Java 运行时错误代码 CSO7160E	884
EGL Java 运行时错误代码 CSO7161E	884
EGL Java 运行时错误代码 CSO7162E	884
EGL Java 运行时错误代码 CSO7163E	884
EGL Java 运行时错误代码 CSO7164E	885
EGL Java 运行时错误代码 CSO7165E	885
EGL Java 运行时错误代码 CSO7166E	885
EGL Java 运行时错误代码 CSO7360E	885
EGL Java 运行时错误代码 CSO7361E	886
EGL Java 运行时错误代码 CSO7488E	886

EGL Java 运行时错误代码 VGJ1219E	978
EGL Java 运行时错误代码 VGJ1220E	978
EGL Java 运行时错误代码 VGJ1221E	978
EGL Java 运行时错误代码 VGJ1222E	979
EGL Java 运行时错误代码 VGJ1223E	979
EGL Java 运行时错误代码 VGJ1224E	979
EGL Java 运行时错误代码 VGJ1225E	979
EGL Java 运行时错误代码 VGJ1226E	980
EGL Java 运行时错误代码 VGJ1227E	980
EGL Java 运行时错误代码 VGJ1228E	980
EGL Java 运行时错误代码 VGJ1229E	980
EGL Java 运行时错误代码 VGJ1290E	981
EGL Java 运行时错误代码 VGJ1301E	981
EGL Java 运行时错误代码 VGJ1302E	981
EGL Java 运行时错误代码 VGJ1303E	981
EGL Java 运行时错误代码 VGJ1304E	982
EGL Java 运行时错误代码 VGJ1305E	982
EGL Java 运行时错误代码 VGJ1306E	982
EGL Java 运行时错误代码 VGJ1401E	982
EGL Java 运行时错误代码 VGJ1402E	983
EGL Java 运行时错误代码 VGJ1403E	983
EGL Java 运行时错误代码 VGJ1404E	983

EGL Java 运行时错误代码 VGJ1405E	983
EGL Java 运行时错误代码 VGJ1406E	984
EGL Java 运行时错误代码 VGJ1407E	984
EGL Java 运行时错误代码 VGJ1408E	984
EGL Java 运行时错误代码 VGJ1409E	984
EGL Java 运行时错误代码 VGJ1410E	984
EGL Java 运行时错误代码 VGJ1411E	985
EGL Java 运行时错误代码 VGJ1412E	985
EGL Java 运行时错误代码 VGJ1414E	985
EGL Java 运行时错误代码 VGJ1415E	985
EGL Java 运行时错误代码 VGJ1416E	986
EGL Java 运行时错误代码 VGJ1417E	986
EGL Java 运行时错误代码 VGJ1419E	986
EGL Java 运行时错误代码 VGJ9900E	986
EGL Java 运行时错误代码 VGJ9901E	987

附录. 声明	989
编程接口信息	991
商标和服务标记.	991

索引	993
---------------------	------------

概述

EGL 简介

企业生成语言（EGL）是一个开发环境和一种编程语言，它使您能够快速编写功能全面的应用程序，并使您能够自由地将注意力放在代码所解决的业务问题上而不是软件技术上。例如，可以使用类似的 I/O 语句来访问不同类型的外部数据存储，而无论那些数据存储是文件、关系数据库还是消息队列。Java™ 和 J2EE 的详细信息对您也是隐藏起来的，因此，即使您只具有最少的 Web 技术经验，您也能够将企业数据传递至浏览器。

在编写 EGL 程序之后，您可以生成它以创建 Java 源代码；接着，EGL 将编译输出以生成可执行的对象。EGL 也可以提供下列服务：

- 将源代码放在开发平台外部的部署平台上
- 在部署平台上编译源代码
- 将状态信息从部署平台发送至开发平台，因此您可以检查结果

EGL 甚至能够生成可以简化可执行对象的最终部署的输出。

可以很容易地将为一个目标平台编写的 EGL 程序转换为用于另一个平台。其好处在于，您可以进行编码以响应当前平台需求，而任何将来迁移的许多详细信息已经为您作了处理。EGL 也可以从同一个源生成应用程序系统的多个部件。

相关概念

- 第 7 页的『开发过程』
- 第 13 页的『EGL 项目、包和文件』
- 第 483 页的『生成的输出』
- 第 16 页的『部件』
- 第 8 页的『运行时配置』

相关任务

- 第 116 页的『创建 EGL Web 项目』

相关参考

- 第 441 页的『EGL 编辑器』
- 第 448 页的『EGL 源格式』

EGL 6.0.0.1 中的新增内容

版本 6.0.0.1 包括下列更改：

- EGL 表单编辑器提供用于创建文本和打印表单的图形用户界面。
- 目标环境包括 HP-UX 和 Solaris。EGL 为这些平台提供 32 位和 64 位支持，并且已经为 AIX 添加了 64 位支持。
- EGL 调试器包括下列更改：
 - 允许您调试基于 consoleUI 的应用程序

- 允许在调试会话期间使用 EBCDIC 代码页来表示字符和数字数据
- 该语言较为灵活:
 - 系统变量 **SysVar.sqlCode** 和 **SysVar.sqlState** 是可修改的
 - 数组下标和子串下标可以包括数字表达式，只要这些表达式不包括函数
 - 如果返回值的类型在表达式中有效，可从数字、文本或逻辑表达式调用返回值的任何函数
 - 如果返回值和参数类型在赋值时是可兼容的，可将返回值的任何函数用作具有修饰符 **in** 的函数参数的自变量
 - 如果自变量和参数类型在赋值时是可兼容的，可将任何 EGL 系统变量作为具有修饰符 **in** 的任何函数参数的自变量传递
 - 任何可修改 EGL 系统变量可作为具有修饰符 **out**（如果自变量和参数类型在赋值时是可兼容的）或者 **inOut**（如果自变量和参数类型在引用时是可兼容的）的函数参数的自变量传递
- 文档现在标识每个 EGL 系统函数中的每个参数的访问修饰符（**in**、**out** 或 **inOut**）；并描述引用和赋值兼容性
- 以下是新提供的系统函数:
 - **MathLib.stringAsDecimal** 接受字符值（如 "98.6"）并返回类型为 DECIMAL 的等效值。
 - **MathLib.stringAsFloat** 接受字符值（如 "98.6"）并返回类型为 FLOAT 的等效值。
 - **MathLib.stringAsInt** 接受字符值（如 "98"）并返回类型为 BIGINT 的等效值。
 - **SysLib.conditionAsInt** 接受逻辑表达式（如 *myVar* == 6），如果表达式求值为 true，则返回 1，如果表达式求值为 false，则返回 0。
 - **SysLib.startLog** 用来打开错误日志。每当程序调用 **SysLib.errorLog** 时，就会将文本写入到该日志中。
 - **SysLib.errorLog** 将文本复制到由系统函数 **SysLib.startLog** 启动的错误日志中。
 - 支持 consoleUI 的新函数:
 - **ConsoleLib.currentArrayCount** 返回与当前活动表单相关联的动态数组中的元素数目。
 - **ConsoleLib.setCurrentArrayCount** 指定绑定至屏幕上的 arrayDictionary 的动态数组中的行数。
 - **ConsoleLib.hideAllMenuItems** 隐藏当前显示菜单中的所有 menuItem
 - **ConsoleLib.showAllMenuItems** 显示当前显示菜单中的所有 menuItem
- 该产品附带包括 Informix 4GL 转换工具
- VAGen 迁移工具已更改，它现在能够进行更有效率的迁移

相关概念

第 11 页的『有关 EGL 的其它信息的来源』

EGL 6.0 iFix 中的新增内容

注: EGL 提供服务以帮助您将旧的代码转换为与 EGL 6.0 iFix 配合使用的代码:

- 如果使用 6.0 之前的版本的 EGL 来创建基于 JavaServer Faces 的 Web 应用程序，则在工作台中执行下列操作：
 1. 单击帮助 > **Rational** 帮助
 2. 在帮助系统的搜索文本框中，至少输入以下文字的开头部分：迁移 Web 项目中的 *JavaServer Faces* 资源。
 3. 单击“执行”
 4. 单击迁移 Web 项目中的 *JavaServer Faces* 资源并遵循该主题中的指示
- 有关从 EGL 6.0 或之前版本迁移代码的其它详细信息，请参阅将 *EGL* 代码迁移至 *EGL 6.0 iFix*。
- 如果要从 Informix® 4GL 或 VisualAge® Generator 迁移代码，请参阅有关 *EGL* 的其它信息的来源。

版本 6.0 iFix 表示 EGL 语句的显著升级：

- 引入了 EGL 报告处理程序，它包含在执行 JasperReports 设计文件期间的不同时段调用的定制函数。从每个函数返回的数据包括在输出报告中，可以 PDF、XML、文本或 HTML 格式呈现。该技术改进了 Informix 4GL 中提供的报告功能。
- 引入了 EGL 控制台用户界面，这是用于创建基于字符的界面的一种技术，它允许用户与 EGL 生成的 Java 程序之间通过击键直接交互。该技术改进了 Informix 4GL 中提供的动态用户界面。
- 增加了代码开发的灵活性：
 - 允许声明新的变量类型：
 - 一种引用变量，它不包含业务数据但指向这类数据。
 - 一种变量，它包含或引用大量数据；具体地说，是引用二进制大对象（BLOB）或字符大对象（CLOB）。
 - 一种字符串变量，它指的是长度在运行时会变化的 Unicode 字符串。
 - ANY 类型的变量，它可以包含任何基本类型的业务数据。
 - 允许您在表达式中加入函数调用。
 - 允许您在不了解记录或该记录中的字段的大小或其它特征的开发时知识的情况下引用记录。每个字段本身可以引用记录。
 - 扩展了对动态数组的支持，现在动态数组可以具有多维。
 - 引入了两种新的数据收集：
 - 一个字典，它由一组键 - 值条目组成。可以在运行时添加、删除和检索这些条目，并且给定条目中的值可以是任何类型。
 - 一个 arrayDictionary，它由一组一维数组组成，每个数组可以是任何类型。可通过检索所有数组中相同编号的元素来访问 arrayDictionary 的内容。
 - 扩展用于以下各种用途的系统函数的数目：
 - 改进用户定义的 Java 运行时属性的日期时间处理、运行时消息处理和检索。
 - 支持与报告、控制台用户界面、BLOB 和 CLOB 有关的新功能。
 - 为异常处理、数据初始化和 DLL 访问提供更好的支持。
- 提供用于创建 EGL 报告处理程序的新向导。
- 允许您定制 Web 页面模板以便与“数据部件和页面”向导配合使用，这就迅速提供了用于访问单个关系数据库的 Web 应用程序。
- 允许您创建代码以反映与空处理和数据库落实有关的 Informix 4GL 运行时行为。

相关概念

第 94 页的『EGL 至 EGL 迁移』

第 11 页的『有关 EGL 的其它信息的来源』

『EGL V6.0 中的新增内容』

EGL V6.0 中的新增内容

注: 如果使用先前版本的 EGL 来创建基于 JavaServer Faces 的 Web 应用程序, 则在工作台中执行下列操作:

1. 单击帮助 > **Rational** 帮助
2. 在帮助系统的搜索文本框中, 至少输入以下文字的开头部分: *迁移 Web 项目中的 JavaServer Faces 资源*。
3. 单击“执行”
4. 单击*迁移 Web 项目中的 JavaServer Faces 资源*并遵循该主题中的指示

版本 6.0 增强了 EGL 语言的功能:

- 处理关系数据库的能力得到改进:
 - 新增向导使您能够迅速完成以下任务:
 - 直接从关系数据库表创建数据部件
 - 创建一些 Web 应用程序, 这些应用程序可从这类表中创建、读取、更新和删除表行
 - 以下是新提供的系统函数:
 - **sysLib.loadTable** 从文件装入信息并将其插入到关系数据库表中。
 - **sysLib.unloadTable** 从关系数据库表卸装信息并将其插入到文件中。
 - 如果要生成 Java 代码, 可通过在游标中导航至下一行 (总是满足条件)、导航至第一行、最后一行、上一行或当前行来访问 SQL 数据库行; 或者通过在游标中指定绝对位置或相对位置来访问 SQL 数据库行。
 - **forEach** 语句使您能够很轻松地在 SQL 结果集的各行间循环。
 - **freeSQL** 语句释放与动态预编译 SQL 语句相关联的所有资源, 关闭与该 SQL 语句相关联的任何打开游标。
- 处理字符串的能力得到改进:
 - 可在文本表达式中指定子串, 如以下示例中所示:

```
myItem01 = "1234567890";  
  
// myItem02 = "567"  
myItem02 = myItem01[5:7];
```
 - 可在文本文字中指定退格、换页或制表符
 - 可对下列任意两种模式类型比较字符串:
 - SQL 类型模式, 它包括 LIKE 关键字。下面是一个示例:

```
// variable myVar01 is the string expression  
// whose contents will be compared to a like criterion  
myVar01 = "abcdef";
```

```
// the next logical expression evaluates to "true"
if (myVar01 like "a_c%")
;
end
```

- 正则表达式模式。下面是一个示例:

```
// variable myVar01 is the string expression
// whose contents will be compared to a match criterion
myVar01 = "abcdef";

// the next logical expression evaluates to "true"

if (myVar01 matches "a?c*")
;
end
```

- 可使用这些文本格式系统函数:

strLib.characterAsInt

将字符串转换为整数字符串

strLib.clip

删除返回的字符串的结束位置的结尾空格和空值

strLib.formatNumber

返回格式字符串形式的数字

strLib.integerAsChar

将整数字符串转换为字符串

strLib.lowercase

将字符串中的所有大写值转换为小写值

strLib.spaces

返回指定长度的字符串。

strLib.upperCase

将字符串中的所有小写值转换为大写值。

- 可声明新类型的变量和结构项。

新增数字类型如下所示:

FLOAT

表示 8 字节区域, 该区域存储最多 16 位有效数字的双精度浮点数

MONEY

表示货币金额, 该金额存储为最多 32 位有效数字的定点小数位

SMALLFLOAT

表示 4 字节区域, 该区域最多存储 8 位有效数字的单精度浮点数

新增日期时间类型如下所示:

DATE

将特定日历日期显示为 8 位单字节数字

INTERVAL

表示时间跨度, 以 1 到 21 个单字节数字表示, 并且与“hhmmss”(表示小时、分钟和秒)之类的掩码相关联

TIME

表示某个时刻，以 6 个单字节数字表示

TIMESTAMP

表示某个时刻，以 1 到 20 个单字节数字表示，并且与“yyyyMMddhh”（表示年、月、日和小时）之类的掩码相关联

- 语法还提供了其它选项:

- 总是可以引用结构项数组的元素，如下所示，但从 iFix 更改的角度来看，最好避免使用此语法:

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

强烈建议使用以下语法:

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

- 可在声明参数、use 语句条目、set 语句条目或变量时使用逗号隔开的标识列表，如下示例所示:

```
myVariable01, myVariable02 myPart;
```

- 在数字表达式中，现在可以通过在值前面加上两个星号（**）来指定指数，所以 8 的立方就是 8**3

- 现在可以指定分别解析为日期、时间、时间戳记或时间间隔的表达式；而且日期算术允许您执行各种任务，如计算两个日期之间的分钟数

- 日期和时间处理可使用下列加法:

- **DateTimeLib.currentTime** 和 **DateTimeLib.currentTimeStamp** 是反映当前时间的系统变量

- 系统提供了新的有关日期（**StrLib.formatDate**）、时间（**StrLib.formatTime**）和时间戳记（**sysLib.TimeStamp**）的格式函数

- 其中每个函数允许您将一系列字符转换为类型为日期时间的项，以便可在日期时间表达式中使用该项:

- **DateTimeLib.dateValue** 返回日期
- **DateTimeLib.timeValue** 返回时间
- **DateTimeLib.timeStampValue** 返回与特定掩码（如“yyyyMMdd”）相关联的时间戳记
- **DateTimeLib.intervalValue** 返回与特定掩码（如“yyyyMMdd”）相关联的时间间隔
- **DateTimeLib.extendDateTimeValue** 接受日期、时间或时间戳记并将其扩展为与特定掩码（如“yyyyMMddmmss”）相关联的项

- 可使用下列新的一般语句:

- **for** 语句包括一个语句块，当测试求值为 true 时会多次循环运行该语句块。循环开始时执行该测试并指示计数器的值是否在指定范围内。

- **continue** 语句将控制权传送到本身包含 **continue** 语句的 **for**、**forEach** 或 **while** 语句的结束位置。包含语句会继续执行还是结束取决于包含语句开始时引导的逻辑测试。

- 可以同步方式（通过发出函数 **sysLib.callCmd**）或异步方式（通过发出函数 **sysLib.startCmd**）运行系统命令。

- 可使用两个新的函数，它们允许您访问循环中的命令行自变量

- **sysLib.callCmdLineArgCount** 返回自变量的数目
- **sysLib.callCmdLineArg** 返回位于自变量列表中的指定位置的自变量
- 现在可指定 **case** 语句，其中每个子句都与一个不同的逻辑表达式相关联。如果使用这一新语法，EGL 运行时将执行与第一个求值为 **true** 的表达式相关联的语句：


```
case when (myVar01 == myVar02)
      conclusion = "okay";
      when (myVar01 == myVar03)
      conclusion = "need to investigate";
      otherwise
      conclusion = "not okay";
end
```
- 可以控制函数参数是仅用于输入、仅用于输出还是同时用于两者；而且您可以通过接受缺省设置（即不受任何限制的“同时用于两者”）来避免进行选择。
- 在下列情况下，现在可以指定比单一项或常量更复杂的日期时间、文本或数字表达式：
 - 通过 **return** 语句指定提供给操作系统的值时
 - 指定在函数调用或程序调用中传递的自变量时；但是，生成时必须知道接收参数的特征
- 现在可以在退出程序时指定复杂数字表达式

开发环境也得到了改进：

- 两个新的功能部件使您能够快速访问部件，即使代码变得很复杂也是如此：
 - “部件引用”视图允许您显示程序、库或 **PageHandler** 引用的 EGL 部件的分层列表；并且可从该列表中访问任何被引用部件
 - EGL 搜索机制允许您指定搜索条件以访问工作空间或一部分项目中的一组部件或变量
- 最后，因为广泛使用了 Web 透视图，所以已经不再使用 EGL Web 透视图了。

相关概念

第 94 页的『EGL 至 EGL 迁移』

第 11 页的『有关 EGL 的其它信息的来源』

第 2 页的『EGL 6.0 iFix 中的新增内容』

开发过程

对 EGL 执行的工作包括下列步骤：

设置

设置工作环境；例如，设置首选项及创建项目。

创建和打开 EGL 文件

开始创建源代码。

声明

创建并指定代码的详细信息。

验证

在不同的时间（如在打开文件时），EGL 查看声明并指示它们在语法上是否正确以及它们的内部（在一定程度上）是否一致。

调试

您可以与内置调试器进行交互以确保代码满足需求。

生成

EGL 验证声明并创建输出，包括源代码。

准备

EGL 准备源代码以生成可执行对象。在某些情况下，此步骤将源代码放置在位于开发平台外部的部署平台上、在部署平台上准备源代码并将结果文件从部署平台发送至开发平台。

直接运行

在某些情况下，只需要右键单击 Java 输出并单击**运行 > Java 应用程序**就可以在“工作台”中直接运行代码。

部署

EGL 生成一些输出，这些输出使得可执行对象的部署更加容易。

相关概念

第 255 页的『EGL 调试器』

第 293 页的『将 Java 代码生成到项目中』

第 1 页的『EGL 简介』

相关任务

第 304 页的『处理生成到目录中的 Java 代码』

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

相关参考

第 441 页的『EGL 编辑器』

第 448 页的『EGL 源格式』

运行时配置

EGL 允许为多个受支持平台中的任何一个平台生成 Java 程序。可以将程序部署在 J2EE 外部，也可以将其部署在下列 J2EE 容器的任何一个容器的上下文中：

- J2EE 应用程序客户机
- J2EE Web 应用程序
- EJB 容器；在这种情况下，还将生成 EJB 会话 bean

另外，EGL 提供了一种方法来定义具有下列特征的 Web 应用程序：

- 将图形页传递至 Web 浏览器
- 能够为潜在的大量用户存储和检索数据
- 嵌入在称为 JavaServer Faces 的基于 Java 的框架中

有关对 Web 应用程序的此项特殊支持的详细信息，请参阅 *PageHandler* 部件。

最后，可以使用 EGL 来生成 Java 包装器，如下一节所述。

使用 Java 包装器

EGL 生成的 Java 包装器是一组类，这些类允许您从非 EGL 生成的 Java 代码（例如，从基于 Struts 或 JSF 的 J2EE Web 应用程序或从非 J2EE Java 程序）调用 EGL 生成的程序。Java 至 EGL 的集成任务如下所示：

1. 生成 Java 包装器类，这些类特定于生成的程序
2. 将那些包装器类合并到非生成的 Java 代码中
3. 从非生成的 Java 代码中调用包装器类方法以进行实际调用并在以下两种格式之间转换数据：
 - Java 使用的数据类型格式
 - 当将数据传递到 EGL 生成的程序以及从 EGL 生成的程序中传递数据时需要的的基本类型格式

有效调用

下表显示对 EGL 生成的代码的有效调用或者来自 EGL 生成的代码的有效调用。

调用对象	被调用对象
J2EE 外部的 Java 类中 EGL 生成的 Java 包装器	EGL 生成的 Java 程序（非 J2EE）
	J2EE 应用程序客户机中 EGL 生成的 Java 程序
	EGL 生成的 EJB 会话 Bean
	VisualAge Generator 生成的 CICS® COBOL 程序
J2EE 应用程序客户机中 EGL 生成的 Java 包装器	EGL 生成的 Java 程序（非 J2EE）
	J2EE 应用程序客户机中 EGL 生成的 Java 程序
	EGL 生成的 EJB 会话 bean
	VisualAge Generator 生成的 CICS COBOL 程序
J2EE Web 应用程序中 EGL 生成的 Java 包装器	EGL 生成的 Java 程序（非 J2EE）
	J2EE 应用程序客户机中 EGL 生成的 Java 程序
	同一个 J2EE Web 应用程序中 EGL 生成的 Java 程序
	EGL 生成的 EJB 会话 bean
	VisualAge Generator 生成的 CICS COBOL 程序

调用对象	被调用对象
J2EE 外部的 EGL 生成的 Java 程序	EGL 生成的 Java 程序 (非 J2EE)
	J2EE 应用程序客户机中 EGL 生成的 Java 程序
	EGL 生成的 EJB 会话 Bean
	VisualAge Generator 生成的 CICS COBOL 程序
	使用 C 或 C++ 编写的非 EGL 生成的程序
	用任何语言编写并在 CICS 下运行的非生成的程序
J2EE 应用程序客户机中 EGL 生成的 Java 程序	EGL 生成的 Java 程序 (非 J2EE)
	J2EE 应用程序客户机中 EGL 生成的 Java 程序
	EGL 生成的 EJB 会话 bean
	EGL 生成的 CICS COBOL 程序
	用任何语言编写并在 CICS 下运行的非生成的程序
	用 C 或 C++ 编写的非生成的程序
J2EE Web 应用程序中 EGL 生成的 Java 程序	EGL 生成的 Java 程序 (非 J2EE)
	J2EE 应用程序客户机中 EGL 生成的 Java 程序
	同一个 J2EE Web 应用程序中 EGL 生成的 Java 程序
	EGL 生成的 EJB 会话 bean
	VisualAge Generator 中生成的 CICS COBOL 程序
	用 C 或 C++ 编写的非生成的程序
EGL 生成的 EJB 会话 bean	EGL 生成的 Java 程序 (非 J2EE)
	J2EE 应用程序客户机中 EGL 生成的 Java 程序
	EGL 生成的 EJB 会话 bean
	VisualAge Generator 生成的 CICS COBOL 程序
	用 C 或 C++ 编写的非生成的程序

有效转移

下表显示面向 EGL 生成的代码的有效转移或者来自 EGL 生成的代码的有效转移。

转移对象	接收对象
J2EE 外部的 EGL 生成的 Java 程序	EGL 生成的 Java 程序 (非 J2EE)
J2EE 应用程序客户机中 EGL 生成的 Java 程序	同一个 J2EE 应用程序客户机中 EGL 生成的 Java 程序
J2EE Web 应用程序中 EGL 生成的 Java 程序	同一个 J2EE Web 应用程序中 EGL 生成的 Java 程序

相关概念

第 483 页的『生成的输出』

第 1 页的『EGL 简介』

第 297 页的『Java 程序、PageHandler 和库』

第 274 页的『Java 包装器』

第 177 页的『PageHandler』

相关任务

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

有关 EGL 的其它信息的来源

以下 Web 站点提供了本文档的最新副本:

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

有关迁移用 VisualAge Generator 编写的源代码的详细信息, 请参阅《*VisualAge Generator 到 EGL 迁移指南*》(vagenmig.pdf 文件), 可在之前提到的 Web 站点或名为安装和迁移的帮助系统章节中找到它。

但是, 建议您访问之前提到的 Web 站点。

但是, 建议您访问 Web 站点。

相关概念

第 1 页的『EGL 简介』

EGL 语言概述

EGL 项目、包和文件

一个 EGL 项目可以不包含源文件夹，也可以包含多个源文件夹；每个源文件夹可以不包含包，也可以包含多个包；每个包可以不包含文件，也可以包含多个文件。每个文件可以不包含部件，也可以包含多个部件。

EGL 项目

EGL 项目的特征由一组属性决定，后面的内容对属性作了描述。在 EGL 项目的上下文中，当您执行特定的任务时（例如，当您保存 EGL 文件或构建文件时），EGL 自动执行验证并解析部件引用。另外，如果您正在使用 PageHandler 部件（它们的输出用来在 Websphere 测试环境中调试 Web 应用程序），则 EGL 自动生成输出，但只在以下情况下才这样：

- 您通过选择下列选项设置了自动构建过程：**窗口 > 首选项 > 工作台 > 资源修改时自动执行构建**
- 您已建立缺省构建描述符来作为首选项或属性

通过在创建新项目时选择 **EGL** 或 **EGL Web** 作为项目类型来构造 EGL 项目。您在完成创建项目的步骤期间指定属性。在完成那些步骤之后，要开始修改您所作的选择，请右键单击项目名，在上下文菜单显示后，单击**属性**。

EGL 属性如下所示：

EGL 源文件夹

一个或多个项目文件夹，这些项目文件夹是项目的包的根文件夹，这些项目文件夹的每一个都是一组子目录。源文件夹对于将 EGL 源文件与 Java 文件分隔开以及将 EGL 源文件存放在 Web 部署目录外部而言非常有用。建议您在所有情况下都指定 EGL 源文件夹；但如果未指定源文件夹，则唯一的源文件夹是项目目录。

此属性的值存储在项目目录中名为 `.eglp` 的文件中，并且保存在用来存储 EGL 文件的存储库（如果有的话）中。

每个 EGL 项目向导都创建一个名为 **EGLSource** 的源文件夹。

EGL 构建路径

项目列表，对于在当前项目中找不到的任何部件，将搜索此列表。

此属性的值存储在项目目录中名为 `.eglp` 的文件中，并且保存在用来存储 EGL 文件的存储库（如果有的话）中。

在以下 `.eglp` 文件示例中，EGLSource 是当前项目中的源文件夹，AnotherProject 是 EGL 路径中的一个项目：

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

AnotherProject 的源文件夹根据该项目中的 `.eglp` 文件确定。

缺省构建描述符

构建描述符，它们使您能够快速生成输出，如工作台中的生成中所述。

包

包是相关源部件的命名集合。在创建构建部件时没有使用任何包。

按照约定，通过将您所在机构的因特网域名倒置作为包名的初始部分来使包名具有唯一性。例如，IBM® 域名为 `ibm.com`，而 EGL 包以“`com.ibm`”开头。通过使用此约定，就可以在在一定程度上确保您所在机构开发的 Web 程序的名称不会与另一机构开发的程序的名称重复，并且这些程序可以安装在同一台服务器上而不可能导致名称冲突。

给定的包的文件夹由包名标识，包名是由句点（.）隔开的标识序列，如以下示例所示：

```
com.mycom.mypack
```

每个标识都与 EGL 源文件夹下的某个子文件夹相对应。例如，`com.mycom.mypack` 的目录结构为 `\com\mycom\mypack`，源文件存储在最底层的文件夹中；在此示例中，文件存储在 `mypack` 中。如果工作空间是 `c:\myWorkspace`，如果项目是 `new.project`，并且，如果源文件夹是 `EGLSource`，则该包的路径如下所示：

```
c:\myWorkspace\new.project\EGLSource\com\mycom\mypack
```

EGL 文件中的部件全都属于同一个包。文件的 `package` 语句（如果有的话）指定了那个包的名称。如果未指定 `package` 语句，则部件直接存储在源文件夹中，并且被称为位于缺省包中。由于缺省包中的文件不能被其它包或项目中的部件共享，因此建议您始终指定 `package` 语句。

不能在同一个包中定义两个具有相同标识的部件。强烈建议您避免在不同的项目或不同的文件夹下使用相同的包名。

生成的 Java 输出的包与 EGL 文件包相同。

EGL 文件

每个 EGL 文件都属于下列其中一个类别：

源文件 EGL 源文件（扩展名为 `.egl`）包含逻辑、数据和用户界面部件，并且是以 EGL 源格式编写的。

下列每个可生成部件都可以变换为可编译单元：

- `DataTable`
- `FormGroup`
- `Handler`（报告处理程序的基础）
- `Library`
- `PageHandler`
- `Program`

EGL 源文件可以不包含子部件，也可以包含多个不可生成部件，但不能包含多个可生成部件。可生成部件（如果有的话）必须位于文件的顶层，并且必须与文件同名。

构建文件

EGL 构建文件（扩展名为 .eglbld）包含任意数目的构建部件，并且是使用可扩展标记语言（XML）以 EGL 构建文件格式编写的。您可以查看相关的 DTD，它位于以下目录中：

```
installationDir\egl\eclipse\plugins\  
com.ibm.etools.egl_version
```

installationDir

产品安装目录，如 C:\Program Files\IBM\Rational\SPD\6.0。如果在安装您要使用的产品之前安装了 Rational® Developer 产品并将它保留下来，则需要指定在之前安装中使用的目录。

version

插件的已安装版本；例如，6.0.0

文件名（如 egl_wssd_6_0.dtd）以字母 *egl* 和下划线开头。字符 *wssd* 指的是 Rational Web Developer 和 Rational Application Developer；字符 *wsed* 指的是 Rational Application Developer for z/OS®；而字符 *wdsc* 指的是 Rational Application Developer for iSeries™。

在将部件添加至文件之后，可以使用存储库来维护更改历史记录。

建议

本节提供有关设置开发项目的建议。

关于构建描述符

小组项目应该指定一位人员作为构建描述符开发者。该人员的任务如下所示：

- 为源代码开发者创建构建描述符。
- 将那些构建描述符放在与源代码项目分开的项目中；并将那个独立的项目放在存储库中，或者通过其它一些方法使之可用
- 请源代码开发者在他们的项目中设置**缺省构建描述符**属性，以使该属性引用适当的构建描述符
- 如果有一小部分构建描述符选项（如用户标识和密码）随源代码开发者的不同而有所变化，则请每位源代码开发者执行下列操作：
 - 编写一个使用 **nextBuildDescriptor** 选项来指向组构建描述符的个人构建描述符
 - 请源代码开发者在他们的文件、文件夹或包中设置**缺省构建描述符**属性，以使该属性引用个人构建描述符。由于项目级别属性和其它项目信息一样处于存储库的控制之下，所以他们不在项目级别指定属性。

有关其它信息，请参阅 *构建描述符部件*。

关于包

关于包，我们有以下建议：

- 不要在不同的项目或源目录中使用相同的包名
- 不要使用缺省包

部件指定

对于部件，许多建议指的是良好的习惯，而不是硬性要求。即使是可选的建议也应该采纳，除非您有很好的理由不这么做：

- 其中一项要求是将 JSP 与它们的相关 PageHandler 放在同一个项目中。
- 如果不可生成部件（如记录部件）仅由一个程序、库或 PageHandler 使用，则将该不可生成部件与使用它的部件放在同一个文件中。
- 如果从同一个包中的不同文件中引用某个部件，则将该部件放在那个包中的独立文件中。
- 如果在单个项目中的各个包之间共享某个部件，则将该部件放在该项目中的独立包中。
- 将完全不相关的应用程序的代码放在不同的项目中。项目是在本地目录结构与存储库之间传送代码的单元。应该将项目结构设计成使开发者能够将必须装入开发系统的代码量降至最低。
- 对项目、包和文件的命名方式应该能够反映它们所包含的部件的用途。
- 如果您的过程强调开发者的代码所有权，则不要将不同所有者的部件指定到同一个文件中。
- 在清楚了解包的用途的情况下将部件指定给包；并按那些部件之间的关系的紧密程度对它们进行分组。

以下区别十分重要：

- 如果将一个部件从同一个包中的一个文件移至另一文件，则不会要求更改其它文件中的 import 语句。
- 如果将一个部件从一个包移至另一个包，则可能要求在引用所移动的部件的每个文件中添加或更改 import 语句。

相关概念

第 267 页的『构建描述符部件』

第 300 页的『工作台中的生成』

第 20 页的『对部件的引用』

第 30 页的『Import』

第 1 页的『EGL 简介』

『部件』

相关参考

第 346 页的『EGL 构建文件格式』

第 448 页的『EGL 源格式』

第 80 页的『EGL 语句』

部件

EGL 文件包含一组部件，每个部件都是一个命名的离散单元。一些部件（如程序）属于可生成部件；每个部件都是可编译单元的基础。可生成部件必须与包含该部件的 EGL 源文件同名。

EGL 源文件（扩展名为 .egl）可以不包含可生成部件，也可以包含一个可生成部件，并且可以不包含其它部件，也可以包含多个其它部件。

部件还可以按下方式进行分类：

• 逻辑部件定义用 EGL 过程语言编写的运行时序列：

- 不可生成部件函数是基本逻辑单元。其它的每种逻辑部件都可以包括函数。
- 可定义两种类型的程序中的任何一种，它们根据界面类型的不同而有所变化。每一个都是可生成部件：
 - 基本程序避免与用户交互或限制与特定种类的基于字符的界面进行交互。在这种情况下，界面技术的工作方式如下：
 - 在命令窗口中显示输出；并且
 - 允许用户与程序直接交互，每一次击键都可能会定义让程序处理的独立事件。

有关这种类型的界面的详细信息，请参阅控制台用户界面。

- *textUI* 程序通过以下方式与用户交互：

- 在命令窗口中显示一组字段；并且
- 仅当用户按提交键时，才接受用户的字段输入。

可将任一类型的程序定义为主程序。这种类型的程序以下列任一方式启动：

- 由用户启动
- 由调用之外的程序转移启动
- 直接由操作系统进程启动

并且，可以将任何一种类型的程序声明为被调用程序，该类程序只能被调用。

有关主程序和被调用程序的运行时部署的其它详细信息，请参阅运行时配置。

- *PageHandler* 是一个可生成部件，用来控制用户与 Web 页面之间的交互。
 - 类型为 *JasperReport* 的处理程序是一个可生成部件，它包含在执行 *JasperReports* 设计文件期间分不同时间调用的定制函数。从每个函数返回的数据包括在输出报告中，可以 PDF、XML、文本或 HTML 格式呈现。
 - 库也是可生成部件；它是可供程序、*pageHandler* 和其它库使用的共享函数和变量的集合。
- 数据部件定义可供程序使用的数据结构。

下列几种数据部件可用作变量声明中的类型：

- *DataItem* 部件包含有关最基本的数据种类的信息。这些部件类似于系统范围的数据字典中的条目，每个部件包括有关数据大小、类型、格式规则、输入验证规则以及显示建议的详细信息。定义 *DataItem* 部件一次，就可以将它用作任意数目的基本变量或记录字段的基础。

DataItem 部件使您能够很方便地从基本类型创建变量。例如，考虑下面的 *myStringPart* 定义，它是类型为字符串的 *DataItem* 部件：

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

开发函数时，可以声明类型为 `MyStringPart` 的变量：

```
myString MyStringPart;
```

下面的声明与先前的声明的作用相同：

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

就像显示的那样，`DataItem` 部件的名称只是具有特定属性设置的基本类型的别名。

- 记录部件是复杂数据的基础。类型为记录部件的变量包括字段。每个字段可以基于下列任何一种类型：

- 基本类型，如 `STRING`
- `DataItem` 部件
- 固定记录部件（如下所述）
- 另一个记录部件
- 任何先前种类的数组

每个字段也可以是字典或 `ArrayDictionary`（如后所述）；或者是一组字典或一组 `ArrayDictionary`。

基于记录部件的变量称为记录，记录中的数据长度在运行时可能会变化。

可使用记录部件来创建变量以进行一般处理或访问关系数据库。

- 固定记录部件是属于固定长度的复杂数据的基础。类型为固定记录部件的变量包括字段，每个字段可以是下列任何类型：

- 基本类型，如 `CHAR`
- `dataItem` 部件

每个字段可以是具有子结构的。例如，可按如下所示定义指定电话号码的字段：

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

尽管您可以将固定记录部件用于各种类型的处理，但它们最好的用途还是用于 `VSAM` 文件、`MQSeries`[®] 消息队列和其它顺序文件的 I/O 操作上。

在一定程度上，`EGL` 支持固定记录部件以允许它们与之前的产品（如 `VisualAge Generator`）兼容。尽管可将固定记录用于访问关系数据库或一般处理，但还是建议您避免将固定记录用于这些用途。

- 字典部件是一直可用的；不用定义它。基于字典部件的变量可能包括键及其相关值的集合，您可以在运行时添加和除去键和值条目。
- `ArrayDictionary` 部件是一直可用的；不用定义它。基于 `ArrayDictionary` 部件的变量允许您通过检索每个数组中编号相同的元素来访问一系列数组。以此方式检索的元素集合本身就是字典，每个数组名被视作与数组元素中的值配对的键。

`ArrayDictionary` 在与控制台用户界面中描述的显示技术相关联时特别有用。

另一数据部件是 `DataTable`，它被视作变量而不是变量类型。`DataTable` 是可由多个程序共享的可生成部件。它包含一系列行和列；每个单元格中包括一个基本值；它被视作运行单元的全局变量（在大多数情况下）。

- 用户界面部件描述在固定字体屏幕和打印表单中向用户显示的数据的布局。用户界面部件可在不同的上下文中使用，并且具有下列类型：
 - 子类型为 *ConsoleForm* 的记录部件是一种数据结构，在 consoleUI 技术的上下文中显示给用户。与其它记录部件相同，每一个这样的记录部件都被用作一个或多个变量的类型；但在此情况下，每个变量被称为控制台表单而不是记录。ConsoleUI 技术还包括为您定义的可用作变量基础的其它部件；有关详细信息，请参阅控制台用户界面。
 - 表单也是一种显示给用户的数据结构。一种类型的表单在 textUI 程序中组织发送至屏幕的数据，另一种类型的表单在任何类型的应用程序中组织发送至打印机的数据。

每个表单包括类似固定记录的固定内部结构；但表单不能包括子结构。

仅当表单由 *FormGroup* 包括或引用时，该表单才对程序、PageHandler 或库可用（如下所述）。

- *FormGroup* 部件是文本和打印表单的集合，它是一个可生成部件。一个程序只能包括一个用于大多数用途的 *formGroup* 以及一个用于与帮助相关的输出的 *formGroup*。可以将同一个表单包括在多个 *FormGroup* 中。

尽管必须在特定于程序的 use 语句中指定访问权，但 *FormGroup* 中的表单对于程序是全局的。这些表单可作为变量引用。

使用 Page Designer 来创建 Web 用户界面，前者构建 JSP 文件并使该文件与 EGL pageHandler 相关联。对于通过 Web 来与用户进行交互的应用程序，JSP 文件替换了用户界面部件的角色。

- 构建部件是在 EGL 构建文件（扩展名为 .eglbld）中定义的，它用来定义各种处理特征：
 - 构建描述符部件控制生成过程并指示在该过程期间读取其它哪些控制部件。
 - 链接选项部件提供有关生成的程序如何转移至其它程序的详细信息。此部件中的信息在生成时、测试时和运行时使用。
 - 资源关联部件将 EGL 记录与访问特定目标平台上的文件所需要的信息相关联；此部件中的信息在生成时、测试时和运行时使用。

固定记录、DataTable 或表单（不管是文本还是打印）包括固定结构。该结构由一系列字段构成，每个字段的大小和类型在生成时已知；如果是 DataTable 或固定记录，则字段可以是具有子结构的。

相关概念

第 79 页的『ArrayDictionary』

第 267 页的『构建描述符部件』

第 400 页的『与 VisualAge Generator 的兼容性』

第 163 页的『控制台用户界面』

第 121 页的『DataItem 部件』

第 75 页的『字典』

第 13 页的『EGL 项目、包和文件』

第 123 页的『固定记录部件』

第 130 页的『函数部件』

第 30 页的『 Import 』
第 1 页的『 EGL 简介 』

第 282 页的『 链接选项部件 』
第 128 页的『 程序部件 』
第 122 页的『 记录部件 』
『 对部件的引用 』
第 53 页的『 引用 EGL 中的变量 』
第 277 页的『 资源关联和文件类型 』
第 8 页的『 运行时配置 』
第 23 页的『 固定结构 』
第 25 页的『 Typedef 』
第 171 页的『 Web 支持 』

相关参考

第 346 页的『 EGL 构建文件格式 』
第 441 页的『 EGL 编辑器 』
第 448 页的『 EGL 源格式 』
第 80 页的『 EGL 语句 』
第 31 页的『 基本类型 』

对部件的引用

本节描述一组规则，这些规则确定 EGL 如何标识名称所引用的部件。在下列情况下，这些规则非常重要：

- 一个函数调用另一个函数
- 一个非函数部件（例如 dataItem 部件）引用验证器函数
- 一个部件在结构项或变量的声明中充当 typedef（格式模型）
- 一个部件在使用声明中引用另一个部件
- 一个构建部件引用另一个构建部件

第二组规则确定 EGL 如何解析变量引用。有关详细信息，请参阅对变量和常量的引用。

基本可视性规则

最简单的情况是，您在一个包中一个接一个地定义部件，而不在一个部件中声明另一个部件。以下列表省略了许多详细信息，但显示了处于同一层次级别的一系列部件：

```
Function: Function01
Function: Function02
Function: Function03
Record: Record01
```

同一级别的各部件可以相互使用。例如，Function01 可以调用其它一个或两个函数；而 Record01 可以用作三个函数的每一个中的变量的 typedef。

在大多数情况下，一个部件不能嵌套另一个部件。例外情况如下所示：

- 程序、库或 pageHandler 可以嵌套函数，但即便如此，包含也必须是直接的；一个函数不能嵌套另一个函数
- 表单组可以嵌套表单

下面是一个具有被嵌套部件的示例:

```
Program: Program01
  Function: Function01
    Function: Function02
  Function: Function03
Record: Record01
```

顶层部件可供包中的所有其它部件使用。但是, 被嵌套部件 (Function01 和 Function02) 只能供包中的一部分部件使用:

- 它们可以相互使用。
- 它们可供嵌套部件以及嵌套部件在运行时使用的函数使用。例如, 如果 Function01 调用 Function03, 则由于在 Program01 中使用了 Function03, 所以 Function03 可以调用 Function02。

最后, 如果代码包含文本或打印表单, 则需要使用声明才能访问包含那些表单的表单组。当访问数据表或库时, 也需要使用声明。有关其它信息, 请参阅[使用声明](#)。

其它可视性规则

大多数开发工作会在多个包之间共享部件。下列规则有效:

- 只要被访问部件具有下列特征, 文件中的任何部件都可以引用其它包中的部件:
 - 是顶层部件
 - 未被声明为 *private*
 - 与引用部件位于同一个项目中, 或者位于引用项目的 EGL 构建路径中列示的项目中

可以按照下列方式提供访问:

- 可以用包名来对部件名进行限定, 在这种情况下, 在源文件中不需要 `import` 语句。例如, 如果包名是 `my.package`, 部件名是 `myPart`, 则可以按如下方式引用该部件:

```
my.package.myPart
```
- 可以使用 `import` 语句, 这有下列优点:
 - `import` 语句使您有可能避免对导入的部件的名称进行限定, 除非必须使用包名才能避免模糊引用。
 - `import` 语句提供了一种方法来记载在源代码中使用了哪些包。

部件名解析

要解析部件引用, EGL 执行搜索, 此搜索包括一个到多个步骤。下列描述对于每个步骤都是适用的:

- 如果找到具有唯一名称的部件, 则搜索成功结束
- 如果找到两个同名的部件, 则搜索会由于错误而结束。

下列情况也是有可能的:

- 用包名对部件引用进行限定; 在这种情况下, 搜索始终只包括一个步骤
- 未用包名对部件引用进行限定, 并且该部件引用不是函数调用
- 未用包名对部件引用进行限定, 并且该部件引用是函数调用

下列情况基本上是不重要的, 但它们可能适用于最后两种情况的任何一种:

- 引用函数中的属性 **containerContextDependent** 可能设置为 *yes*。通过设置该属性，可以扩展用来解析引用的名称空间，如 *containerContextDependent* 所述。
- 如果其中一个函数对程序或 PageHandler 可视，并且名称与 EGL 系统函数的名称完全相同，则引用该函数，而不是引用系统函数。

指定了包名时的部件名解析： 如上所述，在引用部件时可以指定包名，如 *my.package.myPart* 的示例所示。将考虑当前项目，并考虑 EGL 构建路径中列示的任何项目。

如果该引用来自于同一个包中的部件，则下列描述是适用的：

- 包名有效，但不是必需的
- 即使部件被声明为 *private*，也会解析该部件名

未指定包名时的部件名解析（函数调用除外）： 如果一个部件引用除函数以外的部件，并且未指定包名，则按搜索顺序执行的步骤如下所示：

1. 搜索这样的部件：这个部件和引用该部件的部件嵌套在同一容器中。
2. 搜索引用部件所在的文件中显式导入的部件。将考虑当前项目，并考虑 EGL 构建路径中列示的任何项目。

在这种情况下，每个 *import* 语句都显式地引用特定包中的特定部件。这种显式类型 *import* 语句中命名的部件将覆盖当前包中的同名部件。

如果在两个不同的项目中有名称完全相同的包，则给定的显式类型 *import* 语句将使用 EGL 构建路径来执行“最先找到”搜索，即，在找到必需部件时停止。（该部件对于给定项目中的包来说必须是唯一的。）在两个不同项目中存在同名的包并不是错误，但会造成混淆，因此不建议这么做。

如果有两个指定同一部件的显式类型 *import* 语句，则会发生错误。

3. 搜索与引用部件位于同一个包中的顶层部件。将考虑当前项目，并考虑 EGL 构建路径中列示的任何项目。查找两个同名的部件将导致错误。
4. 搜索导入的其它部件。将考虑当前项目，并考虑 EGL 构建路径中列示的任何项目。

在这种情况下，每个 *import* 语句都将引用给定包中的所有部件，这种语句称为通配 *import* 语句。

如果在两个不同的项目中有名称完全相同的包，则给定的通配 *import* 语句将使用 EGL 构建路径来执行“最先找到”搜索，即，在找到必需部件时停止。（该部件对于给定项目中的包来说必须是唯一的。）

如果多个通配 *import* 语句检索同名的部件，则会发生错误。

未指定包名时的函数调用： 如果一个部件调用函数，并且未指定包名，则按搜索顺序执行的步骤如下所示：

1. 搜索这样的函数：它们和调用它们的程序嵌套在同一个容器中。
2. 搜索位于容器的使用声明中指定的库中的函数。
3. 仅对在生成时包括在容器中的函数继续进行搜索。（要包括除了那些嵌套在同一个容器中或者位于库中的函数以外的函数，请将容器属性 **includeReferencedFunctions** 设置为 *yes*。）

对包括的函数执行的搜索是按以下方式进行的:

- a. 搜索容器所在的文件中显式导入的部件。将考虑当前项目, 并考虑 EGL 构建路径中列示的任何项目。

在这种情况下, 每个 `import` 语句都显式地引用特定包中的特定部件。这种显式类型 `import` 语句中命名的部件将覆盖当前包中的同名部件。

如果在两个不同的项目中有名称完全相同的包, 则给定的显式类型 `import` 语句将使用 EGL 构建路径来执行“最先找到”搜索, 即, 在找到必需函数时停止。(该函数对于给定项目中的包来说必须是唯一的。)在两个不同项目中存在同名的包并不是错误, 但会造成混淆, 因此不建议这么做。

如果有两个指定同一部件的显式类型 `import` 语句, 则会发生错误。

- b. 搜索与容器位于同一个包中的顶层函数。将考虑当前项目, 并考虑 EGL 构建路径中列示的任何项目。如果搜索找到两个同名的部件, 则会发生错误。
- c. 搜索导入的其它部件。将考虑当前项目, 并考虑 EGL 构建路径中列示的任何项目。

在这种情况下, 每个 `import` 语句都将引用给定包中的所有部件, 这种语句称为通配 `import` 语句。

如果在两个不同的项目中有名称完全相同的包, 则给定的通配 `import` 语句将使用 EGL 构建路径来执行“最先找到”搜索, 即, 在找到必需部件时停止。(该部件对于给定项目中的包来说必须是唯一的。)

如果多个通配 `import` 语句检索同名的部件, 则会发生错误。

程序调用

当在 `call` 语句或 `transfer` 语句上调用程序时, 调用程序的自变量列表必须与被调用程序的参数列表匹配。自变量和参数不匹配将导致错误。

相关概念

第 13 页的『EGL 项目、包和文件』

第 30 页的『Import』

第 1 页的『EGL 简介』

第 16 页的『部件』

第 53 页的『引用 EGL 中的变量』

相关参考

第 425 页的『containerContextDependent』

第 435 页的『EGL 构建路径和 eglpath』

第 441 页的『EGL 编辑器』

第 448 页的『EGL 源格式』

第 875 页的『使用声明』

固定结构

固定结构建立文本表单、打印表单、`dataTable` 或固定记录部件的格式; 它由一系列字段(每个字段描述基本内存位置)或一组内存位置组成, 如以下示例所示:

```

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);

```

如先前示例中所述，可直接在定义中定义所有字段。或者，也可以指示整个或部分结构与另一个固定记录部件中的结构相同；有关详细信息，请参阅 *Typedef*。

对字段的访问是基于带有点语法的变量名和一系列字段名。如果声明记录 *myRecord* 包含上一示例中给出的结构，则下列每个标识都引用一个内存区：

```

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

```

基本结构字段没有下级结构字段，并且按照下列其中一种方式描述内存区：

- 通过指定长度和基本类型，如以上示例所示；或者
- 通过指向 *dataItem* 部件的声明，如 *Typedef* 中所述。

如先前所示，固定结构中的字段可以具有下级字段。考虑下一个示例：

```

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

```

在定义上级结构字段（如 *topMost*）时，有几种选择：

- 如果未指定长度或基本类型，则上级结构字段具有 **CHAR** 类型，并且 EGL 会计算长度。例如，*topMost* 的基本类型是 **CHAR**，长度为 4。
- 如果指定基本类型，但未指定长度，则 EGL 根据下级结构项的特征计算长度
- 如果同时指定长度和基本类型，则长度必须反映为下级结构字段提供的空间；否则，会发生错误

注： 固定结构字段的基本类型确定每个长度单位的字节数；有关详细信息，请参阅 *基本类型*。

每个基本结构字段都具有一系列属性，这些属性或者是缺省属性，或者是在结构字段中指定的。（结构字段可以引用本身具有属性的 *dataItem* 部件。）有关详细信息，请参阅 *EGL 属性与覆盖概述*。

相关概念

- 第 121 页的『*DataItem* 部件』
- 第 123 页的『固定记录部件』
- 第 59 页的『EGL 属性概述』
- 第 16 页的『部件』
- 第 53 页的『引用 EGL 中的变量』
- 第 25 页的『*Typedef*』

相关参考

- 第 430 页的『数据初始化』

第 448 页的『EGL 源格式』

第 31 页的『基本类型』

第 61 页的『SQL 项属性』

Typedef

类型定义 (typedef) 是用作格式模型的一个部件。使用 typedef 机制的原因如下所示:

- 要标识变量的特征
- 要重用部件声明
- 要强制格式约定
- 要说明数据的含义

typedef 通常标识抽象分组。例如, 可以声明一个名为 *address* 的记录部件, 并将信息划分成 *streetAddress1*、*streetAddress2* 和 *city*。如果个人记录包含结构项 *workAddress* 和 *homeAddress*, 则每一个结构项都可以指向名为 *address* 的记录部件的格式。这样使用 typedef 可以确保地址的格式相同。

在本页描述的规则集中, 可在声明一个部件或在声明变量时指向另一个部件的格式。

当声明部件时, 不需要将部件用作 typedef, 但是您可能想这样做, 正如在稍后显示的示例中所示。并且, 当声明具有数据项特征的变量时, 不需要使用 typedef; 或者, 可以指定变量的所有特征, 而不引用部件。

在声明比数据项更复杂的变量时, typedef 始终有效。例如, 如果声明名为 **myRecord** 的变量并指向名为 **myRecordPart** 的部件的格式, 则 EGL 将根据该部件建立所声明的变量的模型。如果指向名为 **myRecordPart02** 的部件的格式, 则变量将称为 **myRecord**, 但它具有名为 **myRecordPart02** 的部件的所有特征。

下表以及下列各节提供了有关不同上下文中的 typedef 的详细信息。

指向 typedef 的条目	typedef 可以引用的部件类型
函数参数或其它函数变量	记录部件或 dataItem 部件
程序参数	dataItem 部件、表单部件和记录部件
程序变量 (非参数)	dataItem 部件和记录部件
结构项	dataItem 部件和记录部件

DataItem 部件作为 typedef

在下列情况下, 可以将 dataItem 部件用作 typedef:

- 声明变量或参数时
- 声明结构项 (结构项是记录部件、表单部件或 dataTable 部件的子单元) 时

下列规则适用:

- 如果一个结构项是列示在同一声明中的其它结构项的父代, 则该结构项只可指向 dataItem 部件的格式, 如以下示例所示:

```
DataItem myPart CHAR(20) end
```

```
Record myRecordPart type basicRecord  
  10 mySI myPart; // myPart acts as a typedef
```

```

    20 a CHAR(10);
    20 b CHAR(10);
end

```

以上记录部件等同于以下声明:

```

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

```

- 不能将 `dataItem` 部件用作 `typedef` 并同时指定指向该 `typedef` 的实体的长度和基本类型, 如以下示例所示:

```

DataItem myPart HEX(20) end

// NOT valid because mySI has a primitive type
// and points to the format of a part (to myPart, in this case)
Record myRecordPart type basicRecord
  10 mySI CHAR(20) myPart;
end

```

- 不引用记录部件的变量声明将指向 `dataItem` 部件的格式或者具有基本特征。(程序参数也可以引用表单部件。)但是, `dataItem` 部件不能指向另一个 `dataItem` 部件或任何其它部件的格式。
- SQL 记录部件只能将下列类型的部件用作 `typedef`:
 - 另一个 SQL 记录部件
 - `dataItem` 部件

记录部件作为 `typedef`

在下列情况下, 可以将记录部件用作 `typedef`:

- 当声明结构项时
- 当声明变量 (包括参数) 时, 在这种情况下, 变量按照下列方式反映 `typedef`:
 - 格式
 - 记录类型 (例如, `indexedRecord` 或 `serialRecord`)
 - 属性值 (例如, `file` 属性的值)

当声明指向另一部件格式的结构项时, 您指定 `typedef` 是否添加一层层次结构, 如下所示。

下列规则适用:

- 当使用结构项以便于重用, 记录部件可以是 `typedef`:

```

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

```

第二个记录部件等同于以下声明:

```

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

```

如果结构项使用先前的语法来指向结构部件的格式, 则 EGL 会向包含结构项的结构部件添加一层结构。为此, 上一个示例中的内部结构具有结构项层次结构, 并且 *person* 与 *streetAddress1* 处于不同的层次级别。

- 在某些情况下, 您会想要在结构中采用平面布局; 作为用于访问关系数据库的 I/O 对象的 SQL 记录必须具有这样的布局:
 - 在上一个示例中, 如果用单词 **embed** 来替代记录部件的结构项名 (在这种情况下为 *homeAddress*), 并且在该单词后面是充当 typedef 的记录部件的名称 (在这种情况下为 *address*), 则部件声明看起来类似于:

```

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

```

记录部件的内部结构现在是平面布局:

```

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

```

使用单词 **embed** 来替换结构项名的唯一原因是为了避免添加一层层次结构。由单词 **embed** 标识的结构项具有下列限制:

- 可以指向记录部件的格式, 但是不指向 `dataItem` 部件
- 不能指定数组或者包括基本类型规范
- 接下来, 考虑当在两个记录中声明完全相同的结构时记录部件是 typedef 的情况:

```

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; // accepts the structure of common,
               // not the properties
end

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

```

最后两个记录部件等同于下列声明:

```

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

```

- 可以在声明一系列结构项时将记录部件多次用作 `typedef`。此重用是很有意义的，例如，如果您正在声明包含家庭地址和工作地址的个人记录，这就会很有用。基本记录可以在结构中的两个位置提供相同的格式:

```

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

```

此记录部件等同于以下声明:

```

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

```

- 不能将记录部件用作 typedef 并同时指定指向该 typedef 的实体的长度和基本类型, 如以下示例所示:

```

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

```

```

// not valid because myFirst has a
// primitive type and points to the format of a part
Record myStruct02 type serialRecord
{
  fileName = "myFile"
}
  10 myFirst HEX(40) myTypedef;
end

```

但是, 请参照以下实例:

```

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

```

```

Record myStruct02 type basicRecord
  10 myFirst myTypedef;
end

```

第二个结构等同于以下声明:

```

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

```

缺省情况下, 具有下级结构项的任何结构项的基本类型均为 CHAR, 并且该结构项的长度就是下级结构项表示的字节数, 而不管这些结构项的基本类型。有关其它详细信息, 请参阅“结构”。

- 对于 SQL 记录, 下列限制有效:

- 如果一个 SQL 记录部件将另一个 SQL 记录部件用作 typedef, 则该 typedef 提供的每个项都包含 4 个字节的前缀。但是, 如果非 SQL 记录将 SQL 记录部件用作 typedef, 则不包括前缀。有关背景知识信息, 请参阅 *SQL 记录内部结构*。
- SQL 记录部件只能将下列类型的部件用作 typedef:
 - 另一个 SQL 记录部件
 - dataItem 部件
- 最后, 结构和结构项都不能作为 typedef

表单作为 typedef

仅当声明程序参数时才可以将表单部件用作 typedef。

相关概念

第 121 页的『DataItem 部件』

第 142 页的『表单部件』

第 1 页的『EGL 简介』

第 122 页的『记录部件』

第 23 页的『固定结构』

相关任务

第 127 页的『创建 EGL 程序部件』

相关参考

第 80 页的『EGL 语句』

第 682 页的『SQL 记录内部结构』

Import

import 语句标识一组部件, 这些部件位于指定的包中 (对于 EGL 源文件), 或者位于指定的一组文件中 (对于 EGL 构建文件)。包含 import 语句的文件可以引用所导入的部件, 就象是那些部件与该文件位于同一个包中一样。

背景知识

如果一个公用部件位于除当前包以外的包中, 但未在 import 语句中标识该包, 则需要代码中用包名 (例如 my.pkg) 对部件名 (例如 myPart) 进行限定, 如以下示例所示:

```
my.pkg.myPart
```

但是, 如果在 import 语句中标识了该部件, 则可以在代码中去掉包名。在这种情况下, 未限定的包名 (如 myPart) 已足够了。

有关使用 import 语句来解析部件名的情况的描述, 请参阅[对部件的引用](#)。

Import 语句的格式

在 EGL 文件中, import 语句的语法如下所示:

```
import packageName.partSelection;
```

packageName

标识要在其中进行搜索的包的名称。此名称必须是完整的名称。

partSelection

这是部件名或星号 (*)。星号表示选择包中的所有部件。

构建文件中的 `import` 语句标识其它构建文件，那些构建文件的部件可以由正在导入的文件中的部件引用。在构建文件中，`import` 语句跟随在 `<EGL>` 标记后面，并且每个语句都具有以下语法：

```
<import file=filePath.eglbuild>
```

filePath

标识要导入的文件的路径和名称。如果要指定路径，则下列描述是适用的：

- 文件路径位于同一个项目中的任何源目录中，或者位于 EGL 路径中的任何其它项目中
- 各限定符由斜线 (/) 隔开

可以指定星号 (*) 作为文件名或作为文件名的最后一个字符。如果使用星号，则 EGL 导入所有具有下列特征的 `.eglbuild` 文件：

- 位于指定的文件路径中。
- 名称以星号前面的字符开头。（如果星号前面没有字符，则选择目录路径中的所有构建文件。）

文件扩展名 `.eglbuild` 是可选的。

相关概念

第 13 页的『EGL 项目、包和文件』

第 1 页的『EGL 简介』

第 16 页的『部件』

第 20 页的『对部件的引用』

相关任务

第 290 页的『编辑 EGL 构建路径』

基本类型

每种 EGL 基本类型都表现内存区的特征。有三种基本类型：字符、数字和日期时间。

- 字符类型如下所示：
 - *CHAR* 指的是单字节字符。
 - *DBCHAR* 指的是双字节字符。dbchar 替换了 DBCS，后者是 EGL V5 中的一种基本类型。
 - *MBCHAR* 指的是多字节字符，这是单字节字符与双字节字符的组合。mbchar 替换了 MIX，后者是 EGL V5 中的一种基本类型。
 - *STRING* 指的是变长字段，其中的双字节字符符合 Unicode Consortium 开发的 UTF-16 编码标准。
 - *UNICODE* 指的是固定字段，其中的双字节字符符合 Unicode Consortium 开发的 UTF-16 编码标准。
 - *HEX* 指的是十六进制字符。
- 日期时间类型如下所示：
 - *DATE* 指的是具有 8 位单字节固定长度的特定日历日期。

- *INTERVAL* 指的是时间跨度，长度可以是 2 位到 27 位单字节。
- *TIME* 指的是某个时刻，具有 6 位单字节固定长度。
- *TIMESTAMP* 指的是当前时间，长度可以是 2 位到 20 位单字节。
- 大对象类型如下所示：
 - *BLOB* 指的是长度从 1 字节到 2G 字节的二进制大对象。
 - *CLOB* 指的是长度从 1 字节到 2G 字节的字符大对象。
- 数字类型如下所示：
 - *BIGINT* 指的是用于存储多达 18 位的整数的 8 字节区域。此类型等同于长度为 8 并且不带小数位的 *BIN* 类型。
 - *BIN* 指的是二进制数字。
 - *DECIMAL* 指的是符号由最右字节后半部分中的十六进制 C（表示正数）或十六进制 D（表示负数）表示的压缩十进制字符。*DECIMAL* 替换了 *PACK*，后者是 EGL V5.0 中的一种基本类型。
 - *FLOAT* 指的是 8 字节区域，该区域存储最多 16 位有效数字的双精度浮点数
 - *INT* 指的是存储最多 9 位的整数的 4 字节区域。此类型等同于长度为 4 并且不带小数位的 *BIN* 类型。
 - *MONEY* 指的是存储为 *DECIMAL* 值的当前金额。
 - *NUM* 指的是符号由最右字节前半部分中特定于符号的十六进制值表示的数字字符。对于 ASCII，该值为 3（表示正数）和 7（表示负数）；对于 EBCDIC，该值为 F（表示正数）和 D（表示负数）。
 - *NUMC* 指的是符号由最右字节前半部分中特定于符号的十六进制值表示的数字字符。对于 ASCII，该值为 3（表示正数）和 7（表示负数）；对于 EBCDIC，该值为 F（表示正数）和 C（表示负数）。
 - *PACF* 指的是符号由最右字节后半部分中的十六进制 F（表示正数）或十六进制 D（表示负数）表示的压缩十进制字符。
 - *SMALLFLOAT* 指的是 4 字节区域，该区域存储最多 8 位有效数字的单精度浮点数。
 - *SMALLINT* 指的是用于存储多达 4 位的整数的 2 字节区域。此类型等同于长度为 2 并且不带小数位的 *BIN* 类型。

任何定点数字类型的字段的内部表示都与整数表示相同（即使指定了小数点也是如此）。例如，表示为 12.34 与表示为 1234 是相同的。同样，货币符号不会与类型为 *MONEY* 的字段存储在一起。

当您与 DB2[®] 进行交互（直接或通过 JDBC）时，定点数字中的最大位数最多为 31。

类型为 *ANY* 的变量接收指定给该变量的值的类型，如主题 *ANY* 中所述。

在声明时，您指定表现下列每一个值的特征的基本类型：

- 函数返回的值
- 字段中的值，项是按名称引用并包含单个值的内存区

其它实体也有基本类型：

- 系统变量具有特定于该字段的基本类型（通常为 *NUM*）
- 字符文字具有下列类型之一：

- CHAR (如果文字只包含单字节字符的话)
- DBCHAR (如果文字只包含双字节字符集中的双字节字符的话)
- MBCHAR (如果文字包含单字节字符与双字节字符的组合的话)
- 不支持 UNICODE 类型的字符文字。

每种基本类型都是在单独的页上描述的; 可以在涉及赋值、逻辑表达式、函数调用和 call 语句的页上获取其它详细信息。

后面各节将讨论以下主题:

- 声明时的基本类型
- 不同数字类型的相对效率

声明时的基本类型

请参照以下声明:

```
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
```

如上所示, 必须在声明这些实体时指定基本类型:

- 基本变量
- 没有子结构的结构字段

可指定具有子结构的结构字段 (如 *address*) 的基本类型。如果未能指定这样的结构字段的基本类型, 但在代码中又引用了该结构字段, 则产品作出以下假定:

- 假定基本类型为 CHAR, 即使下级结构字段为另一种类型
- 假定长度为下级结构字段中的字节数

不同数字类型的相对效率

EGL 支持类型 DECIMAL、NUM、NUMC 和 PACF, 因此可以更容易地使用旧应用程序所使用的文件和数据库。建议在新的开发中使用 BIN 类型的字段或者使用等效的整数类型 (BIGINT、INT 或 SMALLINT); 原因是对于那些类型的字段, 计算效率最高。使用长度为 2 并且不带小数位的 BIN 类型 (SMALLINT 类型的等效类型) 字段时效率最高。

在计算、赋值和比较中, 不带小数位的类型为 NUM 的字段比具有小数位的类型为 NUM 的字段效率高。

带有类型为 DECIMAL、NUM、NUMC 和 PACF 的字段计算效率相同。

相关概念

第 121 页的『DataItem 部件』

第 122 页的『记录部件』
第 53 页的『引用 EGL 中的变量』
第 23 页的『固定结构』

相关参考

『ANY』
第 340 页的『赋值』
第 46 页的『BIN 和整数类型』
第 513 页的『call』
第 35 页的『CHAR』
第 38 页的『DATE』
第 35 页的『DBCHAR』
第 46 页的『DECIMAL』
第 86 页的『异常处理』
第 47 页的『FLOAT』
第 473 页的『函数调用』
第 36 页的『HEX』
第 38 页的『INTERVAL』
第 454 页的『逻辑表达式』
第 36 页的『MBCHAR』
第 47 页的『MONEY』
第 47 页的『NUM』
第 48 页的『NUMC』
第 461 页的『数字表达式』
第 613 页的『运算符和优先顺序』
第 48 页的『PACF』
第 49 页的『SMALLFLOAT』
第 61 页的『SQL 项属性』
第 37 页的『STRING』
第 462 页的『文本表达式』
第 40 页的『TIME』
第 40 页的『TIMESTAMP』
第 37 页的『UNICODE』

ANY

类型为 ANY 的变量接收指定给该变量的值的类型。该值可以是基本类型（如 INT），也可以是基于用作类型的数据部件的变量。该值不能是表单或 dataTable。

请参照以下示例:

```
myInt INT = 1;
myString STRING = "EGL";

myAny01, myAny02 any;

// myAny01 receives the value 1 and the type INT
myAny01 = myInt;

// myAny02 receives the value "EGL" and the type STRING
myAny02 = myString;
```

```
// The next statement is
// NOT VALID because a variable of type INT
// is being assigned to a variable of type STRING
myAny02 = myAny01;
```

以无效方式组合类型的操作只有在运行时才会检测出来，并且会导致程序终止。这些操作包括对不兼容类型的字段赋值、将自变量值传递给不兼容类型的参数或者将不兼容的值合并到一个表达式中。

文字的类型由该文字的值暗示：

- 加引号的字符串的类型为 `STRING`
- 4 位或少于 4 位的整数的类型为 `SMALLINT`
- 5 到 8 位的整数的类型为 `INT`
- 9 到 18 位的整数的类型为 `BIGINT`
- 包括小数点的数字的类型为 `NUM`

在引用类型为 `ANY` 的变量时，访问始终是动态的。固定结构（`dataTable`、文本表单、打印表单或固定记录）中不能包括类型为 `ANY` 的字段。

相关参考

第 31 页的『基本类型』

字符类型

CHAR

类型为 `CHAR` 的项被解释为单字节字符序列。长度既反映字符数又反映字节数，并且范围是 1 至 32767。

工作站平台（如 Windows® 2000）使用 ASCII 字符集；大型机平台（如 z/OS UNIX® System Services）使用 EBCDIC 字符集。整理顺序的差别通常会导致大于和小于比较在这两种类型的环境中具有不同的结果。

相关参考

第 31 页的『基本类型』

DBCHAR

类型为 `DBCHAR` 的项被解释为双字节字符序列。长度反映了字符数，并且范围是 1 至 16383。要确定字节数，将长度值乘 2。

工作站平台（如 Windows 2000）使用 ASCII 字符集；大型机平台（如 z/OS UNIX System Services）使用 EBCDIC 字符集。整理顺序的差别通常会导致大于和小于比较在这两种类型的环境中具有不同的结果。

DBCS 数据是表意数据，这是显示诸如汉语、日语或韩国语之类的数据所必需的。显示这样的数据需要具有 DBCS 功能的终端设备。

相关参考

第 31 页的『基本类型』

HEX

类型为 HEX 的项被解释为一系列十六进制位（0 至 9、a 至 f 以及 A 至 F），而这些位是作为字符来处理的。长度反映了位数，并且范围是 1 至 65534。要确定字节数，请除以 2。

对于长度为 4 的项，示例值的内部位表示如下所示：

```
// hexadecimal value 04 D2
00000100 11010010

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

类型为 HEX 的项主要用来访问数据类型与另一 EGL 基本类型不匹配的文件或数据库字段。

可以通过使用类型为 CHAR 并且只包含十六进制位范围内的字符的文字来指定十六进制值，如下列示例所示：

```
myHex01 = "ab02";

myHex02 = "123E";
```

可以包括十六进制项来作为逻辑表达式中的操作数，如下列示例所示：

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

不能在算术表达式中包括十六进制项。

相关参考

第 31 页的『基本类型』

MBCHAR

类型为 MBCHAR 的项被解释为单字节字符与双字节字符的组合。长度反映项可以包含的单字节字符数，还反映字节数。长度的范围是 1 至 32767。

工作站平台（如 Windows 2000）使用 ASCII 字符集；大型机平台（如 z/OS UNIX System Services）使用 EBCDIC 字符集。整理顺序的差别通常会导致大于和小于比较在这两种类型的环境中具有不同的结果。

在大型机环境中，如果项中可能出现双字节字符，则必须包括用于 shift-out 和 shift-in 字符的空间：

- 单字节 Shift-out 字符（十六进制值 0E）指示一系列双字节字符的开始
- 单字节 Shift-in 字符（十六进制值 0F）指示该系列的结束

Shift-out 和 Shift-in 字符是在 EBCDIC 到 ASCII 数据转换期间被删除的，并且是在 ASCII 到 EBCDIC 数据转换期间被插入的。如果正在转换变长记录，并且如果当前记录末尾（由记录长度所示）位于类型为 MBCHAR 的结构项内，则将调整记录长度以反映 Shift-out 和 shift-in 字符的插入或删除。

双字节字符数据是表意数据，这是显示诸如汉语、日语或韩国语之类的的数据所必需的。显示这样的数据需要具有双字节字符集功能的终端设备。

相关参考

第 31 页的『基本类型』

STRING

基本类型 STRING 由双字节 UNICODE 字符组成。

可以将字段的值存储在文件或数据库中。如果代码与 DB2 UDB 进行交互，则必须确保 GRAPHIC 数据的代码页是 UNICODE 并且存储数据项值的列具有 SQL 数据类型 GRAPHIC 或 VARGRAPHIC。

有关 Unicode 的详细信息，请访问 Unicode Consortium 的 Web 站点 (www.unicode.org)。

相关参考

第 31 页的『基本类型』

UNICODE

基本类型 UNICODE 提供了一种处理和存储可能由几种人类语言中任何一种表示的文本的方法；但是，必须已从代码外的位置提供了该文本。不支持类型为 UNICODE 的文字。

下列情况适用于类型为 UNICODE 的项：

- 长度反映了字符数，并且范围是 1 至 16383。为这样的项保留的字节数是对长度指定的值的两倍。
- 该项只能被赋予另一个类型为 UNICODE 的项或与该项作比较。
- 所有比较操作都将根据字符在 UTF-16 编码标准中的顺序来比较位值。
- 必要时，EGL 将以 Unicode 空格来填充项。
- 系统字符串函数将项视为单个字节的字符串，这些字节包括添加的 Unicode 空格（如果有的话）。您在那些函数中指定的任何长度都必须以字节为单位而不是以字符为单位。
- 可以将项的值存储在文件或数据库中。如果代码与 DB2 UDB 进行交互，则必须确保 GRAPHIC 数据的代码页是 UNICODE 并且存储数据项值的列具有 SQL 数据类型 GRAPHIC 或 VARGRAPHIC。

有关 Unicode 的详细信息，请访问 Unicode Consortium 的 Web 站点 (www.unicode.org)。

相关参考

第 31 页的『基本类型』

DateTime 类型

DATE

类型为 DATE 的项是一串 8 位单字节数字，反映特定日历日期。

类型 DATE 的格式为 *yyyyMMdd*:

yyyy

表示年份的四位数字。范围是 0000 到 9999。

MM

表示月份的两位数字。范围是 01 到 12。

dd 表示日子的两位数字。范围是 01 到 31，如果代码指定无效日期（如 20050230），将发生错误。

如果该项是用在主机环境中，则示例值的内部十六进制表示如下所示:

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

如果该项是用在使用 ASCII 的 Windows 2000 这样的工作站环境中，则示例值的内部十六进制表示如下所示:

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

类型为 DATE 的项可从关系数据库接收数据，也可以向关系数据库提供数据。

相关参考

第 723 页的『EGL 库 DateTimeLib』

第 453 页的『日期时间表达式』

第 31 页的『基本类型』

第 42 页的『日期、时间和时间戳记格式说明符』

INTERVAL

类型为 INTERVAL 的项是一串 1 到 20 位单字节数字，反映时间间隔，即两个时间点之间之差。每一位的含义由声明该项时指定的掩码确定。

时间间隔可以是正数（从 2005 减去 1980 时）或负数（从 1980 减去 2005 时），该项的开头有一个额外字节，在掩码中不会显示出来。如果记录中有类型为 INTERVAL 的项，则在计算记录的长度和上级项（如果有的话）的长度时，必须将该额外字节计算在内。

可以下列任一格式指定掩码:

- 跨度为月，可包括年份和月份
- 跨度为秒，可包括天、小时、分钟、秒和百分秒

任一情况下，掩码中的每个字符都表示 1 位数字。在跨度为月的格式中，例如，*y* 的组合指示项中包含多少年。如果只需要 3 位数字来表示年数，则在掩码中指定 *yyy*。如果需要最大位数（9）来表示年数，则指定 *yyyyyyyyyy*。

在给定掩码中，第一个字符最多可用 9 次（除非另行声明）；但每个后续种类字符的使用次数会进一步受到限制。

对于跨度为月的格式的掩码，将按顺序提供下列字符：

y 表示时间间隔中的年数的 0 到 9 位数字。

M 表示时间间隔中的月数的 0 到 9 位数字。如果 *M* 不是掩码中的第一个字符，则最多只允许使用两位数字。

缺省掩码为 *yyyyMM*。

对于以跨度为秒的格式的掩码，将按顺序提供下列字符：

d 表示时间间隔中的天数的 0 到 9 位数字。

H 表示时间间隔中的小时数的 0 到 9 位数字。如果 *H* 不是掩码中的第一个字符，则最多只允许使用两位数字。

m 表示时间间隔中的分钟数的 0 到 9 位数字。如果 *m* 不是掩码中的第一个字符，则最多只允许使用两位数字。

s 表示时间间隔中的秒数的 0 到 9 位数字。如果 *s* 不是掩码中的第一个字符，则最多只允许使用两位数字。

f 0 到 6 位数字，每个数字表示秒的尾数，第一位表示十分之几，第二位表示百分之几，依此类推。即使 *f* 是掩码中的第一个字符，最多也只允许使用六位数字。

尽管掩码的开头或结尾可以使用指定种类的零字符，但不能跳过中间的字符。有效掩码包括下列各项：

```
yyyyyyMM
YYYYYY
MM
ddHHmmsffffff
HHmmsfff
mmss
HHmm
```

但下列掩码因为缺少中间字符而无效：

```
// NOT valid
ddmmsffffff
HHsfff
```

如果是缺省掩码 (*yyyyMM*) 起作用，并且该项是用于使用 EBCDIC 的主机环境上，则示例值的内部十六进制表示如下所示：

```
// 100 years, 2 months; the 4E means the value is positive
4E F0 F1 F0 F0 F0 F2
```

```
// 100 years, 2 months; the 60 means the value is negative
60 F0 F1 F0 F0 F0 F2
```

如果是缺省掩码 (*yyyyMM*) 起作用，并且该项是用于使用 ASCII 的 Windows 2000 这样的工作站环境上，则示例值的内部十六进制表示如下所示：

```
// 100 years, 2 months; the 2B means the value is positive
2B 30 31 30 30 30 32
```

```
// 100 years, 2 months; the 2D means the value is negative
2D 30 31 30 30 30 32
```

类型为 INTERVAL 的项属于强类型，所以不能将此类型的项与任何其它类型的项进行比较；也不能将此类型的项与其它类型的项相互赋值。

总之，类型为 INTERVAL 的项不能从关系数据库接收数据，也不能向关系数据库提供数据。

相关参考

第 723 页的『EGL 库 DateTimeLib』

第 453 页的『日期时间表达式』

第 31 页的『基本类型』

第 42 页的『日期、时间和时间戳记格式说明符』

TIME

类型为 TIME 的项是一串 6 位单字节数字，反映特定时刻。

类型 TIME 的格式为 *HHmmss*:

HH

表示小时的两位数字。范围是 00 到 24。

mm

表示小时内的分钟数的两位数字。范围是 00 到 59。

ss 表示分钟内的秒数的两位数字。范围是 00 到 59。

如果该项是用在主机环境上，则示例值的内部十六进制表示如下所示：

```
// 8:40:20 o'clock
F0 F8 F4 F0 F2 F0
```

如果该项是用在使用 ASCII 的 Windows 2000 这样的工作站环境上，则示例值的内部十六进制表示如下所示：

```
// 8:40:20 o'clock
30 38 34 30 32 30
```

类型为 TIME 的项可从关系数据库接收数据，也可以向关系数据库提供数据。

相关参考

第 723 页的『EGL 库 DateTimeLib』

第 453 页的『日期时间表达式』

第 31 页的『基本类型』

第 42 页的『日期、时间和时间戳记格式说明符』

TIMESTAMP

类型为 TIMESTAMP 的项是一串 1 到 20 位单字节数字，反映特定时刻。每一位的含义由声明该项时指定的掩码确定的。

指定掩码时将按顺序提供下列字符：

YYYY

表示年份的四位数字。范围是 0000 到 9999。

MM

表示月份的两位数字。范围是 01 到 12。

dd 表示日子的两位数字。范围是 01 到 31。

HH

表示小时的两位数字。范围是 00 到 23。

mm

表示分钟的两位数字。范围是 00 到 59。

ss 表示秒数的两位数字。范围是 00 到 59。

f 0 到 6 位数字，每个数字表示秒的尾数，第一位表示十分之几，第二位表示百分之几，依此类推。

缺省掩码为 *yyyyMMddHHmmss*。

当您与 DB2（直接或通过 JDBC）进行交互时，必须指定从年份（*yyyy*）到秒数（*ss*）的每个部分。在其它上下文中，以下描述是适用的：

- 掩码的开头或结尾可以使用指定种类的零字符，但不能跳过中间的字符。
- 有效掩码包括下列各项：

```
yyyyMMddHHmmss  
YYYY  
MMss
```

- 下列掩码因为缺少中间字符而无效：

```
// NOT valid  
ddMMssffffff  
HHssff
```

如果是缺省掩码（*yyyyMMddHHmmss*）起作用，并且该项是用于使用 EBCDIC 的主机环境中，则示例值的内部十六进制表示如下所示：

```
// 8:05:10 o'clock on 12 January 2005  
F2 F0 F0 F5 F0 F1 F1 F2 F0 F8 F0 F5 F1 F0
```

如果是缺省掩码（*yyyyMMddHHmmss*）起作用，并且该项是用于使用 ASCII 的 Windows 2000 这样的工作站环境中，则示例值的内部十六进制表示如下所示：

```
// 8:05:10 o'clock on 12 January 2005  
32 30 30 35 30 31 31 32 30 38 30 35 31 30
```

类型为 `TIMESTAMP` 的项可与类型为 `TIMESTAMP` 的项或类型为 `DATE`、`TIME`、`NUM` 或 `CHAR` 的项进行比较（或相互赋值）。但是，如果指定无效的值，则在开发时可能会发生错误。下面是一个示例：

```
// NOT valid because February 30 is not a valid date  
myTS timestamp("yyymdd");  
myTS = "20050230";
```

如果完整掩码的开头缺少字符（例如，如果掩码为“*dd*”），则 EGL 假定高级字符（在此情况下为“*yyyyMM*”）表示当前时刻以符合机器时钟。下列语句导致二月份出现运行时错误：

```
// NOT valid because February 30 is not a date  
myTS timestamp("dd");  
myTS = "30";
```

总之，类型为 `TIMESTAMP` 的项可从关系数据库接收数据，也可以向关系数据库提供数据。

相关参考

- 第 340 页的『赋值』
- 『日期、时间和时间戳记格式说明符』
- 第 453 页的『日期时间表达式』
- 第 723 页的『EGL 库 DateTimeLib』
- 第 454 页的『逻辑表达式』
- 第 31 页的『基本类型』

日期、时间和时间戳记格式说明符

日期、时间和时间戳记的格式是由字母模式指定的，每个字母表示日期或时间的组件。这些字符是区分大小写的，并且从 a 到 z 以及从 A 到 Z 的所有字母将解析为日期或时间的组件。

要在文本未解析为日期或时间的组件的情况下显示日期、时间或时间戳记中的字母，用单引号将字母括起来。要在日期、时间或时间戳记中显示单引号，请使用两个单引号。

下表列示日期、时间或时间戳记模式中的字母及它们的值。

字母	日期或时间组件	类型	示例
G	Era designator	文本	AD
y	年份	年份	1996; 96
M	月份	月份	July; Jul; 07
w	当年的周	数字	27
W	当月的周	数字	2
D	当年的某天	数字	189
d	当月的某天	数字	10
F	星期几	数字	2
E	星期几	文本	Tuesday; Tue
a	AM/PM 标记	文本	PM
H	当天的小时 (0-23)	数字	0
k	当天的小时 (1-24)	数字	24
K	上午或下午的小时 (0-11)	数字	0
h	上午或下午的小时 (1-12)	数字	12
m	小时内的分钟	数字	30
s	分钟内的秒	数字	55
S	毫秒	数字	978
z	时区	常规时区	太平洋标准时间; PST; GMT-08:00
Z	时区	RFC 822 时区	-800
C	世纪	世纪	20; 21

在模式中连续使用的字母的数目确定该组字母的解释和解析的方式。解释取决于字母的类型。同时，解释也取决于该模式是用于定义格式还是解析。以下列表描述字母类型以及不同数目的字母影响解释的方式。

文本 对于格式，如果字母的数目小于 4，则使用完整格式。否则将使用缩写（如果可用的话）。在进行解析时，所有格式都可以接受，不管模式字母的编号如何都是如此。

数字 对于格式，模式字母的数目表示最小位数。可向较短数字添加零以使它们达到指定的长度。对于解析，模式字母的数目会被忽略，除非需要用它来区分两个相邻字段。

年份 对于格式，如果模式字母的数目小于 4，则年份会被截断成 2 位。否则，它将被解释为数字类型。

对于解析，如果模式字母的数目不是 2，则年份会以文字方式解释，不管位数如何都是如此。例如，模式 `MM/dd/yyyy` 指定的值 `01/11/12` 解析为公元 12 年 1 月 11 日。同一模式指定的值 `01/02/3` 或 `01/02/0003` 解析为公元 3 年 1 月 2 日。同样，同一模式指定的值 `01/02/-3` 解析为公元前 4 年 1 月 2 日。

对于解析，如果模式为 `yy`，则解析器会确定相对于当年的完整年份。解析器假定两位年份在处理时间的 80 年前或 20 年后的范围内。例如，如果当年为 2004，则模式 `MM/dd/yy` 指定的值 `01/11/12` 解析为 2012 年 1 月 11 日，而同一模式指定的值 `05/04/64` 解析为 1964 年 5 月 4 日。

月份 如果模式字母的数目为 3 或以上，则月份会被解释为文本类型。否则，它将被解释为数字类型。

常规时区

如果常规时区有名称，它们将被解释为文本类型。对于表示 GMT 偏移值的时区，将使用以下语法：

`GMTOffsetTimeZone = GMT Sign Hours : Minutes`

符号 + 或 -

小时 从 0 到 23 的一位到两位数字。格式与语言环境无关，必须从 Unicode 标准的基本 的基本拉丁语块获取。

分钟 从 00 到 59 的两位数字。格式与语言环境无关，必须从 Unicode 标准的基本 的基本拉丁语块获取。

对于解析，RFC 822 时区也是可以接受的。

RFC 822 时区

对于格式，将使用 RFC 822 4 位时区

`RFC822TimeZone = Sign TwoDigitHours : Minutes`

`TwoDigitHours` 必须是从 00 到 23 的两位数字。其它定义与常规时区类型相同。

对于解析，常规时区也是可以接受的。

世纪 显示为数字类型，将完整年份以 100 求模的值。

下表列示用美国语言环境解释的日期和时间模式的示例。

日期和时间模式	结果
<code>yyyy.MM.dd G 'at' HH:mm:ss z</code>	2001.07.04 AD at 12:08:56 PDT
<code>EEE, MMM d, 'yy</code>	Wed, Jul 4, '01
<code>h:mm a</code>	12:08 PM

日期和时间模式	结果
hh 'o''clock' a, zzzz	12 o'clock PM, Pacific Daylight Time
K:mm a, z	0:08 PM, PDT
yyyyy.MMMMM.dd GGG hh:mm aaa	02001.July.04 AD 12:08 PM
EEE, d MMM yyyy HH:mm:ss Z	Wed, 4 Jul 2001 12:08:56 -0700
yyMMddHHmssZ	010704120856-0700

LOB 类型

CLOB

类型 CLOB 的项表示长度从 1 字节到 2G 字节的字符大对象。

下列陈述适用于类型为 CLOB 的项:

- 它仅被声明为一个项，在 BasicRecords 中不受支持。
- 它可被传递至局部函数和程序调用。大对象参数和相应自变量必须都被声明为同一类型的大对象。
- 它只能被指定给另一个 CLOB 变量。
- 它可以移至另一个 CLOB 变量，这与对 CLOB 变量赋值的结果相同。
- 可以创建类型为 BLOB 的引用变量。
- 它使用 SQLlocator (CLOB)；即，CLOB 包含指向 SQL CLOB 数据而不是数据本身的逻辑指针。
- 在与 SQLRecord 一起使用时，
 - CLOB 将字符大对象表示为数据库中的列。
 - CLOB 在创建它的转换持续期间有效。
- 它不能传递给对远程程序或非 EGL 程序的调用。
- 它不能作为赋值语句或表达式中的操作数引用。

可以将下列函数与 CLOB 配合使用:

- attachClobToFile
- freeClob
- getClobLen
- getStrFromClob
- getSubStrFromClob
- loadClobFromFile
- setClobFromString
- setClobFromStringAtPosition
- truncateClob
- updateClobToFile

相关参考

第 45 页的『BLOB』

第 760 页的『EGL 库 LobLib』
第 761 页的『attachClobToFile()』
第 762 页的『freeClob()』
第 763 页的『getClobLen()』
第 763 页的『getStrFromClob()』
第 764 页的『getSubStrFromClob()』
第 764 页的『loadClobFromFile()』
第 765 页的『setClobFromString()』
第 765 页的『setClobFromStringAtPosition()』
第 766 页的『truncateClob()』
第 767 页的『updateClobToFile()』
第 31 页的『基本类型』

BLOB

类型为 BLOB 的项表示长度从 1 字节到 2G 字节的二进制大对象。

下列陈述适用于类型为 BLOB 的项:

- 它仅被声明为一个项，在 BasicRecords 中不受支持。
- 它可被传递至局部函数和程序调用。大对象参数和相应自变量必须都被声明为同一类型的大对象。
- 它只能被指定给另一个 BLOB 变量。
- 它可以移至另一个 BLOB 变量，这与对 BLOB 变量赋值的结果相同。
- 可以创建类型为 BLOB 的引用变量。
- 它使用 SQLlocator (CLOB)；即，BLOB 包含指向 SQL BLOB 数据而不是数据本身的逻辑指针。
- 在与 SQLRecord 一起使用时，
 - BLOB 将二进制大对象表示为数据库中的列。
 - BLOB 在创建它的转换持续期间有效。
- 它不能传递给对远程程序或非 EGL 程序的调用。
- 它不能作为赋值语句或表达式中的操作数引用。

可以将下列函数与 BLOB 配合使用:

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

相关参考

第 44 页的『CLOB』
第 760 页的『EGL 库 LobLib』
第 761 页的『attachBlobToFile()』
第 762 页的『freeBlob()』
第 762 页的『getBlobLen()』

第 764 页的『loadBlobFromFile()』

第 766 页的『truncateBlob()』

第 767 页的『updateClobToFile()』

第 31 页的『基本类型』

数字类型

BIN 和整数类型

类型为 BIN 的项被解释为二进制值。长度可以是 4、9 或 18 并且以十进制格式反映正数位的数目，包括任何小数位。例如，值 -12.34 能够放在长度为 4 的项中。一个 4 位数字需要 2 个字节；一个 9 位数字需要 4 个字节；一个 18 位数字需要 8 个字节。

对于长度为 4 的项，示例值的内部位表示如下所示：

```
// for decimal 1234, the hexadecimal value is 04 D2:
00000100 11010010

// for decimal -1234, the value is the 2's complement (FB 2E):
11111011 00101110
```

建议您尽可能使用类型为 BIN 而不是其它数字类型的项；例如，在以下情形中：算术操作数或结果、数组下标以及相对记录中的键项。

下列类型等同于 BIN 类型：

- BIGINT 的长度为 18，没有小数位
- INT 的长度为 9，没有小数位
- SMALLINT 的长度为 4，没有小数位

相关参考

第 31 页的『基本类型』

DECIMAL

类型为 DECIMAL 的项是一个数值，它的每一个半字节都是一个十六进制字符，且符号是由最右字节后半部分中的十六进制 D（表示正数）或十六进制 D（表示负数）表示的。

长度反映位数，范围是 1 到 32。

要确定字节数，将长度值加 2，将和除以 2，然后截去结果中的任何小数部分。

对于长度为 4 的项，示例值的内部十六进制表示如下所示：

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

// for decimal -123
00 12 3D

// for decimal 1234
01 23 4C

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

从文件或数据库读取到类型为 DECIMAL 的字段中的负数值可能会以十六进制 B 代替 D; EGL 接受该值, 但将 B 转换为 D。

类型为 DECIMAL 的 DB2 UDB 列的格式与 DECIMAL 类型的主变量的格式相同。

相关参考

第 31 页的『基本类型』

FLOAT

对于最多 16 个有效数字的双精度浮点数, 类型为 FLOAT 的项被解释为二进制值。长度固定为 8 字节。在 EGL 生成的 Java 程序中, 值的范围从 4.9e-324 到 1.7976931348623157e308。

FLOAT 对应下列每个定义:

- 关系数据库管理系统中的 FLOAT 数据类型
- C、C++ 或 Java 中的双精度数据类型

对于浮点值, Java 与主机 COBOL 格式之间的格式转换受 DB2 支持, 但对于主机程序的调用不受支持。

相关参考

第 31 页的『基本类型』

MONEY

类型为 MONEY 的项是数字值, 在很多方面相当于类型为 DECIMAL 的项。对于 MONEY, 缺省长度为 16; 缺省小数位为 2; 最小长度为 2; 货币符号将显示在输出字段中。MONEY 对应 IBM Informix 4GL MONEY 数据类型。

该格式基于变量 defaultMoneyFormat。

相关参考

第 46 页的『DECIMAL』

第 61 页的『定义格式的属性』

第 31 页的『基本类型』

NUM

类型 NUM 的项是一个数值, 它的每个字节都是字符格式的一位, 符号由最右字节前两部分中特定于符号的十六进制值表示。长度既反映位数又反映字节数。长度的范围是 1 至 32。

对于长度为 4 的项, 如果项处于主机环境 (使用 EBCDIC) 中, 则示例值的内部十六进制表示如下所示:

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

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

如果项处于工作站环境 (如 Windows 2000, 它使用 ASCII) 中, 则示例值的内部十六进制表示如下所示:

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

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

相关参考

第 31 页的『基本类型』

NUMC

类型 NUMC 的字段是一个数值，它的每个字节都是字符格式的一位，符号由最右字节前半部分中特定于符号的十六进制值表示。长度既反映位数又反映字节数，并且范围是 1 至 18。

对于长度为 4 的字段，如果字段处于主机环境（使用 EBCDIC）中，则示例值的内部十六进制表示如下所示：

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

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

如果字段处于工作站环境（如 Windows 2000，它使用 ASCII）中，则示例值的内部十六进制表示如下所示：

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

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

相关参考

第 31 页的『基本类型』

PACF

类型为 PACF 的字段是一个数值，它的每一个半字节都是一个十六进制字符，且符号是由最右字节后半部分中的十六进制 F（表示正数）或十六进制 D（表示负数）表示的。长度反映了位数，并且范围是 1 至 18。要确定字节数，将长度值加 2，将和除以 2，然后截去结果中的任何小数部分。

对于长度为 4 的字段，示例值的内部十六进制表示如下所示：

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

// for decimal -123
00 12 3D

// for decimal 1234
01 23 4F

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

从文件或数据库读取到类型为 PACF 的字段中的负数值可能会以十六进制 B 代替 D；EGL 接受该值，但将 B 转换为 D。

相关参考

第 31 页的『基本类型』

SMALLFLOAT

对于最多 8 个有效数字的单精度浮点数，类型为 SMALLFLOAT 的项被解释为二进制值。长度固定为 4 字节内存存储。

在 EGL 生成的 Java 程序中，值的范围从 3.40282347e+38 到 1.40239846e-45。

SMALLFLOAT 对应下列每个定义：

- 关系数据库管理系统中的 SMALLFLOAT 数据类型
- C、C++ 或 Java 中的浮点数据类型

对于浮点值，Java 与主机 COBOL 格式之间的格式转换受 DB2 支持，但对于主机程序的调用不受支持。

相关参考

第 31 页的『基本类型』

声明 EGL 中的变量和常量

可以按照下列方式声明变量：

- 可以让变量基于若干基本类型的其中一种，如以下示例所示：

```
myItem CHAR(10);
```

- 可以让变量基于 dataItem 部件、记录部件或固定记录部件，如以下示例所示：

```
myRecord myRecordPart;
```

- 可以让变量基于字典或 arrayDictionary 的特定配置，如以下示例所示：

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

- 程序或其它可生成部件可访问 dataTable 的字段，dataTable 被视作程序或运行单元的全局变量。如果 dataTable 列示在程序的其中一个使用声明中，可使用较简单的语法来访问这些字段。
- 程序可访问文本或打印表单的字段，该表单被视作程序的全局变量。程序的使用声明中必须包括相关的 formGroup。
- 程序或其它可生成逻辑部件可访问在任何库函数外部声明的库变量。这些变量是运行单元的全局变量。如果该库列示在程序的其中一个使用声明中，可使用较简单的语法来访问这些字段。

通过指定符号 CONST 并后跟常量名称、类型、等号和值来声明常量；指定的值在运行时不能更改。示例如下所示：

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

常量不能在记录或其它复杂结构中。

最后，在单个语句中声明多个变量或常量，用逗号隔开每个标识，如下列示例所示：

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

相关概念

第 20 页的『对部件的引用』

第 16 页的『部件』

第 25 页的『Typedef』

相关参考

第 31 页的『基本类型』

第 875 页的『使用声明』

动态和静态访问

EGL 通过静态或动态访问来解析变量引用：

- **动态访问**工作时，字段名和类型只有在运行时才是已知的。代码通过代码中的值或运行时输入确定名称。

当代码要引用下列任何一项时，动态访问会工作：

- 基本类型为 ANY 的变量。
- 字典中的值字段；该字段的类型为 ANY。
- 记录中的字段，当引导至该字段（从记录到字段到子字段）的关系链中的先前引用使用动态访问时
- 由 EGL 括号语法引用的字段。在这种情况下，字段名不一定遵循标识规则，但可以是 EGL 保留字或可以包括在其它情况下可能无效的空格和其它字符。

有关详细信息，请参阅*动态访问的括号语法*。

- **静态访问**工作时，字段名和类型在生成时已知，并且名称始终符合 EGL 标识的命名约定。在运行时不会使用该名称。

当代码要引用下列任何一项时，静态访问会工作：

- 不在任何容器内并且其类型并非 ANY 的变量
- 固定记录中的字段
- 非固定记录中的字段，当引导至该字段（从变量到字段到子字段）的关系链中的每个引用使用静态访问时

考虑以下示例，在该示例中，字典中的值包括固定记录和非固定记录：

```
// a fixed record part
Record myFixedRecordPart type=serialRecord
{
  fileName = "myFile"
}
10 ID INT;
10 Job CHAR(10);
end

// a record part (not fixed)
```

```

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

```

下列规则适用:

- 对字典值的引用是动态的，并且每个下级引用都是动态的。考虑代码包括下列语句的后果:

```

myDynamicID INT;
myDynamicID = relationship.dynamicRecord.ID;

```

对 `dynamicRecord` 的引用将是动态的，对 `ID` 的引用也将是动态的，并且标识 `ID` 在运行时是可视的。

- 以固定结构开头的引用只能引用该结构内部的内存。在当前示例中，以 `fixedPerson` 开头的引用可以访问固定记录中的字段 `ID` 和 `JOB`，但不能访问任何其它字段。
- 代码可以动态访问固定结构，但同一引用语句不能访问该字段的字段。在当前示例中，以下引用是无效的，原因是标识 `ID` 在运行时不可用:

```

myFixedID INT;

// NOT valid
myFixedID = relationship.fixedRecord.ID;

```

您可以通过声明另一固定记录并对其指定来自字典中的固定记录中的值来处理该问题:

```

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

```

动态访问在逻辑表达式中的赋值（在左边或右边）中是有效的；在语句 **set**、**for** 和 **openUI** 中也是有效的。

相关概念

第 55 页的『动态访问的括号语法』

第 75 页的『字典』
第 128 页的『程序部件』
第 53 页的『引用 EGL 中的变量』
第 25 页的『Typedef』

相关任务

第 49 页的『声明 EGL 中的变量和常量』

相关参考

第 340 页的『赋值』
第 454 页的『逻辑表达式』
第 31 页的『基本类型』
第 579 页的『set』

EGL 中的确定作用域规则和“this”

如果 EGL 部件声明变量或常量，则在声明中使用的标识在整个部件的作用域中（可用）：

- 如果声明在函数中，则标识在该函数的局部作用域中。例如，如果函数 Function01 声明变量 Var01，则 Function01 中的任何代码都可以引用 Var01。标识甚至在声明之前的函数代码中也是可用的。

该数组可作为自变量传递给另一个函数，但原始标识在该函数中不可用。参数名在接收函数中可用，这是因为该参数名是在该位置声明的。

- 如果声明在可生成部件（如程序）中但在任何函数中，则标识在程序全局作用域中，这表示该标识可由该部件调用的任何函数引用。例如，如果程序声明 Var01 并调用 Function01，而 Function01 又调用 Function02，则 Var01 在两个函数的整体范围内可用。

文本或打印表单中的标识对于引用该表单的可生成部件是全局的。这些标识甚至在提供该表单的函数之前的函数中也是可用的。

- 如果声明在库中但在任何函数中，则标识在运行单元作用域中，这表示它对运行单元中的所有代码是全局的。
- dataTable 及其字段的名称可能在程序全局作用域、运行单元作用域或更大的作用域中，这取决于 dataTable 属性的设置和 dataTable 所处的环境。

完全相同的标识不能在同一作用域中。但是，大多数标识引用逻辑上在容器（如记录）中的内存区；在这些情况下代码用封闭容器的名称来限定标识。例如，如果函数变量 myString 在称为 myRecord01 的记录中，代码会将变量作为记录字段引用：

```
myRecord01.myString
```

如果同一个标识在两个作用域中，则对该标识的任何引用是对最内部的作用域的引用，但您可以使用限定符来覆盖该行为：

- 考虑这样一个程序，该程序声明变量 Var01 并调用本身声明同名变量的函数。在函数中针对 Var01 的未限定引用会导致访问局部声明的变量。

要访问程序全局的标识（甚至在局部标识优先的情况下），用关键字 *this* 来限定该标识，如以下示例中所示：

```
this.Var01
```

很少情况下，关键字 *this* 也用于覆盖赋值语句中 `set value` 块的行为。有关详细信息，请参阅 *set value* 块。

- 考虑下列情况：
 - 程序具有用于访问库的使用声明；并且
 - 程序和库分别声明名为 `Var01` 的变量。

如果程序中的函数包括针对 `Var01` 的未限定引用，则函数将访问程序变量。

要在运行单元作用域中访问标识（甚至在另一标识阻止该访问的情况下），用部件名来限定该标识，如以下示例中所示（其中 `myLib` 是库的名称）：

```
myLib.Var01
```

如果库或 `dataTable` 在不同包中并且您未在 `import` 语句中引用该部件，则必须在部件名之前加上包名，如以下示例中所示（其中 `myPkg` 是包名）：

```
myPkg.myLib.Var01
```

包名总是限定部件名，不能直接放在变量或常量标识之前。

最后，如果局部标识与 `dataTable` 或库在不同的包中，则局部标识可以与 `dataTable` 或库名相同。要引用 `dataTable` 或库名，则加上包名。

相关概念

- 第 130 页的『函数部件』
- 第 131 页的『类型为 `basicLibrary` 的库部件』
- 第 131 页的『类型为 `basicLibrary` 的库部件』
- 第 177 页的『`PageHandler`』
- 第 16 页的『部件』
- 第 128 页的『程序部件』
- 第 20 页的『对部件的引用』
- 『引用 EGL 中的变量』
- 第 59 页的『EGL 属性概述』
- 第 23 页的『固定结构』
- 第 25 页的『`Typedef`』

相关任务

- 第 49 页的『声明 EGL 中的变量和常量』

相关参考

- 第 473 页的『函数调用』
- 第 481 页的『EGL 源格式的函数部件』

引用 EGL 中的变量

有关两种内存访问之间的差别的详细信息，请参阅[动态和静态访问](#)。

不管是哪种访问在起作用，EGL 带点语法通常就足够了。考虑下列部件定义，例如：

```
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

```

假定声明名为 *myRecordVar01* 的变量时，函数将记录部件 *myRecordPart01* 用作类型。

要引用字段 *myInt*，按顺序列示下列符号：

- 变量的名称；在此情况下为 *myRecordVar01*
- 句点 (.)
- 字段列表引导出您关心的字段，并且会用句点将标识隔开；例如，
myRecordVar02.myRecordVar03
- 您所关心的字段名称前面有句点；在此情况下为 *.myInt*

如果存在数组，将一再使用同一语法。例如，如果 *myRecordVar03* 被声明为包含三个记录的数组，则使用下列符号来访问该数组的第三个元素中的字段 *myInt*：

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

在此示例中引用字典字段时，带点语法也会起作用。要访问值“Twain”，在赋值语句的右边指定下列字符：

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

如果两个不同记录部件中存在名为 *myDictionary* 的字段，不会造成任何问题，原因是每个同名字段在被引用时仅与它自己的封闭记录相关联。

还可以在库（例如 *myLib*）中使用带点语法来引用常量（如 *myConst*）：

```
myLib.myConstant
```

还有两种其它语法可用：

- 在使用动态访问时，您可能希望将字段名指定为带引号的字符串或类型为 **STRING** 的标识。此功能主要用于添加或检索字典条目（键 - 值对）的情况，在这些情况下：
 - 键是 EGL 保留字或包含在标识中无效的字符（如句点或空格）；或者
 - 您希望使用字符串常量来指定或引用该键。

该语法要求您将变量、常量或文字放在一对方括号（[]）中。填有常量的方括号相当于后跟有效标识的点，您可以同时使用这两种语法。但是，引用的开头必须是标识。

有关示例，请参阅[动态访问的括号语法](#)。

- 您可能希望能够使用缩写语法在固定结构（dataTable、文本表单、打印表单或固定记录）中引用字段。但建议您避免使用此语法，这有利于取得之前提到的完全资格。

仅当将属性 **allowUnqualifiedItemReferences** 设置为 *yes* 时，缩写语法才可能相对于固定结构有效。该属性是可生成逻辑部件（如程序、库和 `pageHandler`）的特征，缺省值是 *no*。

有关详细信息，请参阅[静态访问的缩写语法](#)。

相关概念

第 57 页的『引用固定结构的缩写语法』

『动态访问的括号语法』

第 50 页的『动态和静态访问』

第 441 页的『EGL 中的枚举』

第 130 页的『函数部件』

第 16 页的『部件』

第 128 页的『程序部件』

第 20 页的『对部件的引用』

第 52 页的『EGL 中的确定作用域规则和“this”』

第 23 页的『固定结构』

第 25 页的『Typedef』

相关任务

第 49 页的『声明 EGL 中的变量和常量』

相关参考

第 68 页的『数组』

第 473 页的『函数调用』

第 481 页的『EGL 源格式的函数部件』

第 31 页的『基本类型』

第 875 页的『使用声明』

动态访问的括号语法

只要动态访问有效，您就可以在括号中使用字符串变量、常量或文字来引用字段。每对填有常量的方括号相当于后跟有效标识的点。

尽管在字典声明中指定的所有键都必须符合 EGL 标识的规则，但您可以通过在 EGL 赋值语句中使用方括号语法来指定范围更广的键。方括号语法在下一示例中是必需的，在该示例中，会向字典添加两个条目并且会检索每个条目中的值：

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

row["Record"] = "Reserved word";
row["ibm.com"] = "Think!";

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

如果通过在带点语法中使用标识来引用值，可通过使用相当于标识的字符串在方括号语法中引用同一个值。以下赋值的作用相同：

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

假定您声明了名为 `myRecordVar01` 的记录，该记录包括名为 `myRecordVar02` 的字段，而 `myRecordVar02` 本身是包含先前字典的记录。下面是一个有效引用：

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

对于该引用而言，大多数情况下访问是静态的。如果您要访问字典中的字段，则会使用动态访问。假定这些常量在作用域中，但是：

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

可以按如下所示来编写先前的引用：

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

引用中的第一个符号必须始终是有效标识，但在此情况下，动态访问在该标识之后生效。

可以同时使用带点语法和方括号语法。例如，以下引用相当于先前引用：

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

下面是最后一个示例，考虑带有数组下标的引用：

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

假定这些常量在作用域中：

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

可以下列方式编写先前引用：

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

相关概念

第 57 页的『引用固定结构的缩写语法』

第 50 页的『动态和静态访问』

第 130 页的『函数部件』

第 16 页的『部件』

第 128 页的『程序部件』

第 20 页的『对部件的引用』

第 53 页的『引用 EGL 中的变量』

第 52 页的『EGL 中的确定作用域规则和“this”』

第 23 页的『固定结构』

第 25 页的『Typedef』

相关任务

第 49 页的『声明 EGL 中的变量和常量』

相关参考

第 68 页的『数组』

第 473 页的『函数调用』

第 481 页的『EGL 源格式的函数部件』

第 606 页的『MQ 记录的选项记录』

第 31 页的『基本类型』

第 875 页的『使用声明』

引用固定结构的缩写语法

在 dataTable、文本表单、打印表单或固定记录中引用字段时，下列规则适用：

- 如果要引用容器（如固定记录）中的字段，可以使用普通的带点语法以避免正在引用的内存区出现歧义。例如，请参照以下部件声明：

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

假定声明名为 *myRecordVar* 的变量时，函数将记录部件 *myRecordPart* 用作类型。

对 *myRecordVar* 中的 *myChar* 的有效引用如下所示：

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

该引用被认为是标准的。

- 如果要引用其名称在结构中唯一的字段，可以指定变量名，后跟一个句点，再后跟字段名。对前面的示例的有效引用包括以下符号：

```
myRecordVar.myChar
```

该引用被认为是部分限定的。

不能以任何其它方式来部分限定字段名。例如，不能只包括某些位于变量名与您关注的字段名之间的字段名，也不能消除变量名但却保留您所关注的字段的上级结构字段的任何名称。下列引用对于前面的示例无效：

```
// NOT valid
myRecordVar.myNext.myChar
myRecordVar.myAlmost.myChar
myNext.myChar
myAlmost.myChar
```

- 可以引用某个字段而不必在名称前面加上任何限定符。对前面的示例的有效引用包括下列符号：

```
myChar
myChar02
```

那些引用被认为是未限定的。

- 必须将对结构字段的任何引用限定至能够避免发生歧义的程度。
- 如果结构字段的相关内存区是填充符（一个其名称不太重要的区域），则结构字段的名称可以是一个星号（*）。引用中不能包含星号。请参照以下示例：

```

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

```

如果声明变量 *myRecordVar* 时将该部件用作类型，则可以引用 *myRecordVar.nation* 或 *nation*，但是下列引用是无效的：

```

// NOT valid
myRecordVar.*.streetAddress1
myRecordVar.*.streetAddress2
myRecordVar.*.nation

```

- 当 EGL 尝试解析引用时，首先搜索局部变量的名称，然后搜索同一个函数中用于 I/O 的记录中的结构字段的名称，接着搜索其它局部结构字段的名称，最后搜索程序的全局名称。

考虑这样一种情况：函数既声明了名为 *nation* 的基本变量又声明了指向以下基本记录的变量：

```

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

```

对 *nation* 的未限定引用将引用基本变量，而不引用结构字段。

- 名称搜索不会显示基于程序全局结构字段的程序全局基本变量的任何首选项。考虑这样一种情况：程序既声明了名为 *nation* 的基本变量又声明了指向以下基本记录格式的变量：

```

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

```

由于 *nation* 既可以引用基本变量也可以引用结构字段，所以对 *nation* 的未限定引用失败。可以引用结构字段，但是只通过限定引用来进行。

有关其它规则，请参阅 [数组和使用声明](#)。

相关概念

第 130 页的『函数部件』

第 16 页的『部件』

第 128 页的『程序部件』

第 20 页的『对部件的引用』

第 53 页的『引用 EGL 中的变量』

第 52 页的『EGL 中的确定作用域规则和“this”』

第 23 页的『固定结构』

第 25 页的『Typedef』

相关任务

第 49 页的『声明 EGL 中的变量和常量』

相关参考

第 68 页的『数组』

第 473 页的『函数调用』

第 481 页的『EGL 源格式的函数部件』

第 606 页的『MQ 记录的选项记录』

第 31 页的『基本类型』

第 875 页的『使用声明』

EGL 属性概述

大多数 EGL 部件都具有一组属性，这些属性用于在生成时创建相应的输出。有效的属性集合根据上下文的不同而变化：

- 每个部件类型定义一组属性，这些属性将用于部件类型整体。例如，每个程序部件都有一个称为 **alias** 的属性，它标识可编译单元的名称。

如果部件本身是子类型，将提供其它属性。类型为 **textUI** 的程序具有称为 **alias** 的属性和称为 **inputForm** 的属性。后者标识程序逻辑运行之前显示给用户的文本表单。

- 许多部件类型还会定义一组属性，以供在作为该部件类型组件的任意基本字段中使用。例如，类型为 **SQLRecord** 的记录部件包括一组基本字段，每个字段具有一个列属性，用来标识由该字段访问的 SQL 表列。

DataItem 部件中可用的属性包括在任意上下文中有效的所有基本字段级别属性。例如，考虑这样一种 **DataItem** 部件，它表示一个仅有 9 位数字的标识，并且该标识有时与称为 **SSN** 的关系数据库列相关联：

```
DataItem IDPart CHAR(9)
{
    minInput = 9,        // requires 9 input characters
    isDigits = yes,     // requires digits
    columnName = "SSN" // is related to a column
}
```

可声明类型为 **IDPart** 的变量，如下所示：

```
myVariable IDPart;
```

可在组合部件（如记录部件）中声明该变量，也可以直接在逻辑部件（如程序）中声明该变量。在每种情况下，部件类型确定是否使用给定属性。

在当前示例中，仅当在类型为 **SQLRecord** 的记录中声明该变量时，才使用属性 **columnName**。仅当在用户界面部件（如 **pageHandler**）中声明该变量时，才使用两个验证属性。

- 在某些变量声明中，可覆盖在相关部件定义中指定的属性，但仅当该属性在声明变量的上下文中有用时才可如此：

- 如果是声明基于 `DataItem` 部件的变量，则可以在上下文中进行覆盖。以下语句声明类型为 `SSN` 的 `PageHandler` 字段（按之前定义的那样），但该语句不要求用户输入数字：

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

在此示例中，属性 `minInput` 不受覆盖操作的影响，并且属性 `columnName` 会被忽略。

- 在大多数情况下，不能覆盖组合部件（如记录部件）的属性。
- 在定义固定结构时，可对基本结构字段指定属性，并且可以在声明相关变量时覆盖这些属性。还可以对具有下级结构字段的结构字段指定属性，但在这些情况下，被指定属性将会被忽略，除非属性文档另行声明。
- 在声明基本类型的变量时，可设置在变量声明的上下文中有用的任何基本字段级别属性。

不能在运行时访问属性。例如，在创建基于 `SQL` 记录部件的变量时，您所编写的逻辑不能检索或更改指定给 `tableNames` 属性的名称，该属性标识由该记录访问的 `SQL` 表。即使覆盖变量声明中的属性值，您所编写的逻辑也不能更改在开发时指定的值。

如果不能在运行时访问属性值，则意味着当您指定变量的内容或将变量用作自变量时，该属性值不会随内容一起传送。例如，如果将一个 `SQL` 记录中的数据传送至另一个 `SQL` 记录，则有关目标记录访问哪些 `SQL` 表的规范不会有任何更改。同样，在将 `SQL` 记录传递至 `EGL` 函数时，参数将接收字段内容，但保留在开发时指定的 `SQL` 表规范。

预定义 `EGL` 部件（如 `ConsoleField`）可能包括属性和字段。与属性不同，字段在运行时是可用的。您编写的逻辑可读取字段值并且在许多情况下可以更改字段值。

`set-value` 块是一个代码区域，可在其中设置属性和字段值。有关详细信息，请参阅 `set-value` 块。

相关概念

第 53 页的『引用 `EGL` 中的变量』

第 62 页的『`set-value` 块』

相关参考

第 467 页的『`EGL` 源格式的表单部件』

第 61 页的『`SQL` 项属性』

字段显示属性

`EGL` 字段显示属性指定当目标是命令窗口但不是 `Web` 浏览器时，在屏幕输出中显示字段时有意义的特征。

属性如下所示：

- 第 630 页的『`color`』
- 第 638 页的『`highlight`』
- 第 639 页的『`intensity`』
- 第 646 页的『`outline`』

另外，当目标是打印机或打印文件时，当在可打印输出中显示字段时，下列属性有意义：

- **highlight** 属性（但仅对于 *underline* 和 *noHighLight*）
- **outline** 属性，此属性仅适用于支持双字节字符的设备

字段显示属性不影响从文本表单返回给程序的数据；它们只用于输出。

定义格式的屬性

定义格式的屬性指定一些特征，当在表单或 Web 浏览器中显示数据时，这些特征有意义：

- 第 629 页的『align』
- 第 632 页的『currency』
- 第 632 页的『currencySymbol』
- 第 633 页的『dateFormat』
- 第 637 页的『fillCharacter』
- 第 639 页的『isBoolean』
- 第 642 页的『lineWrap』
- 第 642 页的『lowerCase』
- 第 642 页的『masked』
- 第 646 页的『numericSeparator』
- 第 646 页的『outline』
- 第 650 页的『sign』
- 第 652 页的『timeFormat』
- 第 653 页的『timeStampFormat』
- 第 654 页的『upperCase』
- 第 660 页的『zeroFormat』

相关概念

第 59 页的『EGL 属性概述』

SQL 项属性

SQL 项属性指定当在类型为 `SQLRecord` 的记录中使用项时有意义的特征。但是，由于提供了缺省值，所以不需要指定任何 SQL 项属性。

属性如下所示：

- 第 631 页的『column』
- 第 640 页的『isNullable』
- 第 641 页的『isReadOnly』
- 第 643 页的『maxLen』
- 第 647 页的『persistent』
- 第 650 页的『sqlDataCode』
- 第 651 页的『sqlVariableLen』

验证属性

验证属性限制当用户在文本表单中输入数据时被接受的内容。

属性如下所示:

- 第 637 页的 『 fill 』
- 第 638 页的 『 inputRequired 』
- 第 638 页的 『 inputRequiredMsgKey 』
- 第 639 页的 『 isDecimalDigit 』
- 第 640 页的 『 isHexDigit 』
- 第 643 页的 『 minimumInput 』
- 第 644 页的 『 minimumInputMsgKey 』
- 第 644 页的 『 needsSOSI 』
- 第 654 页的 『 typeChkMsgKey 』
- 第 655 页的 『 validatorDataTable 』
- 第 656 页的 『 validatorDataTableMsgKey 』
- 第 656 页的 『 validatorFunction 』
- 第 657 页的 『 validatorFunctionMsgKey 』
- 第 658 页的 『 validValues 』
- 第 659 页的 『 validValuesMsgKey 』

set-value 块

set-value 块是一个代码区域，可在其中设置属性和字段值。有关背景，请参阅 *EGL 属性概述*。

set-value 块在您采取下列任何操作时可用:

- 定义部件
- 声明变量
- 编写特殊格式的赋值语句
- 编写 **openUI** 语句，如 *openUI* 中所述

在后两种情况下，仅对字段赋值。

注: 固定结构中的字段存在限制。可使用 set-value 块来指定基本字段级别属性的值，但不能使用它设置字段本身的值。

基本情况的 set-value 块

考虑大多数基本情况下适用的规则:

- 每个 set-value 块以左花括号 ({) 开始 (包括用逗号隔开的一系列条目或单个条目) 并以右花括号 (}) 结束
- 这些条目全部使用以下两种格式的其中之一:
 - 每个条目由标识 - 值对组成，如 **inputRequired = yes**; 或者
 - 每个条目包含按位置指定的值，即将连续值赋给了一个数组的连续元素。

在所有情况下，set-value 块在要修改的部件、变量或字段的作用域中。下面用示例对语法中的变化作最好的演示。

第一个示例显示 dataItem 部件，它有两个属性（inputRequired 和 align）：

```
// the scope of the set-value block is myPart
DataItem myPart INT
{
  inputRequired = yes,
  align = left
}
end
```

下一个示例显示基本类型的变量。

```
// the scope is myVariable
myVariable INT
{
  inputRequired = yes,
  align = left
};
```

下一个示例显示了 SQL 记录部件声明，它包括两个记录属性（tableNames 和 keyItems）：

```
// The scope is myRecordPart
Record myRecordPart type SQLRecord
{ tableNames = ["myTable"],
  keyItems = ["myKey"] }
myKey CHAR(10);
myOtherKey CHAR(10);
myContent01 CHAR(60);
myContent02 CHAR(60);
end
```

下列示例显示了一个变量声明，该变量声明将先前部件用作类型，忽略了两个记录属性中的其中一个，并在该记录中设置两个字段：

```
// The scope is myRecord
myRecord myRecordPart
{
  keyItems = ["myOtherKey"],
  myContent01 = "abc",
  myContent02 = "xyz"
};
```

其它示例包括变量声明和赋值语句：

```
// the example shows the only case in which a
// record property can be overridden in a
// variable declaration.
// the scope is myRecord
myRecord myRecordPart {keyItems = ["myOtherKey"]};

// the scope is myInteger, which is an array
myInteger INT[5] {1,2,3,4,5};

// these assignment statements
// have no set-value blocks
myRecord02.myContent01 = "abc";
myRecord02.myContent02 = "xyz";

// this abbreviated assignment statement
// is equivalent to the previous two, and
// the scope is myRecord02
myRecord02
```

```

    {
      myContent01="abc",
      myContent02="xyz"
    };

// This abbreviated assignment statement
// resets the first four elements of the array
// declared earlier
myInteger{6,7,8,9};

```

缩写赋值语句对固定结构中的字段不可用。

字段的字段的 **set-value** 块

在对字段的字段赋值时，使用这样的语法：其中的 `set-value` 块在作用域中，所以这些条目仅修改您所关注的字段。

考虑下列部件定义：

```

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

```

可对任何字段指定属性值，如下所示：

- 为该记录创建 `set-value` 块
- 嵌入一系列字段名以缩小作用域
- 创建特定于字段的 `set-value` 块

指定属性值的语法可采用下列适用于字段 `myInt04` 的三种格式中的任何一种，如以下示例中所示：

```

// dotted syntax, as described in
// References to variables in EGL.
myRecB myBasicRecPart
{
  myRec02.myRec03.myInt04{ align = left }
};

// bracket syntax, as described in
// Bracket syntax for dynamic access.
// You cannot use this syntax to affect
// fields in fixed structures.
myRecC myBasicRecPart
{
  myRec02["myRec03"]["myInt04"]{ align = left }
};

// curly-brace syntax
myRecA myBasicRecPart
{
  myRec02 {myRec03 { myInt04 { align = left }}}
};

```

甚至在复杂情况下，您也可以使用逗号在 `set-value` 块中分开各个条目；但您需要考虑嵌套给定块的级别：

```
// dotted syntax
myRecB myBasicRecPart
{
  myInt01 = 4,
  myInt02 = 5,
  myRec02.myRec03.myInt04{ align = left },
  myRec02.myInt03 = 6
};

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

// curly-brace syntax;
// but this usage is much harder to maintain
myRecA myBasicRecPart
{
  myInt01 = 4,
  myInt02 = 5,
  myRec02
  {
    myRec03
    { myInt04
      { action = label5 }},
    myInt03 = 6
  }
};
```

“this”的用法

在变量声明或赋值语句中，可以有一个容器（如 SQL 记录）来包括与记录属性同名的字段（*keyItems*）。要引用字段而不是属性，使用关键字 **this**，这将为 `set-value` 块或 `set-value` 块中的条目建立正确的作用域。

考虑以下记录声明：

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

以下记录声明首先为属性 `keyItems` 设置值，然后为同名字段设置值：

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

下一节给出数组声明中的另一示例。

set-value 块、数组和数组元素

在声明动态数组时，可指定元素的初始数目，如以下示例中所示：

```
coll ConsoleField[5];
```

set-value 块中的赋值引用类型为 ConsoleField 的每一个初始元素中的属性和预定义字段（可是并非引用以后添加的任何元素）：

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

要对变量声明中的特定元素赋值，创建作用域为该元素的嵌入式 set-value 块。如以下示例中所示，通过将关键字 **this** 与带方括号的下标来指定作用域：

```
// assign values to the second and fourth element
coll ConsoleField[5]
{
  this[2] { color = blue },
  this[4] { color = blue }
};
```

有关关键字 **this** 的另一种用法的详细信息，请参阅 *EGL* 中的确定作用域规则和“*this*”。

可在 set-value 块中的位置条目来对以下任何类型的数组中的连续元素赋值（仅在处理报告或创建控制台表单时才起作用）：

- ConsoleField
- 菜单
- MenuItem
- 提示
- 报告
- ReportData

以下示例可以在 OpenUI 语句中。每个嵌入式 set-value 块的作用域是一个特定数组元素：

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

  // property value is a dynamic array
  [
    new MenuItem
    { name = "Expand",
      labelText = "Expand" },
    new MenuItem
    { name = "Collapse",
      labelText = "Collapse" }
  ]
}
```

其它示例

考虑以下部件：

```

Record Point
  x, y INT;
end

Record Rectangle
  topLeft, bottomRight Point;
end

```

以下代码有效:

```

Function test()
  screen Rectangle
  {
    topLeft{x=1, y=1},
    bottomRight{x=80, y=24}
  };

  // change x, y in code, using a statement
  // that is equivalent to the following code:
  //   screen.topLeft.x = 1;
  //   screen.topLeft.y = 2;
  screen.topLeft{x=1, y=2};
end

```

接下来, 在同一函数中初始化类型为 `Point` 的动态元素数组:

```

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

```

设置数组中的每个元素的值, 然后将第一个元素设置为另一个值:

```

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

```

在先前示例中, 因为数组名有歧义, 所以将使用 `pts[1]` 而不是 `this[1]`。

接下来, 考虑类型为 `Point` 的另一个动态数组:

```

points Point[];

```

因为不存在任何元素, 所以下赋值语句不起作用:

```

points{x=1, y=1};

```

相比之下, 因为引用了特定元素并且该元素不存在, 所以下赋值语句将导致越界异常:

```

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

```

可将元素添加至数组, 然后使用单个语句设置所有元素的值:

```

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

```

相关概念

第 55 页的『动态访问的括号语法』

第 59 页的『EGL 属性概述』

第 53 页的『引用 EGL 中的变量』

第 52 页的『EGL 中的确定作用域规则和“this”』

相关参考

『数组』

第 430 页的『数据初始化』

第 565 页的『openUI』

数组

EGL 支持下列类型的数组:

- 『动态数组』
- 第 71 页的『结构字段数组』

在任一情况下, 最多支持 7 个维。

动态数组

在声明记录、固定记录或基本变量数组时, 该数组的标识与数组中的元素无关:

- 有一组特定于该数组的函数, 它们允许您在运行时增加或减少元素数目。
- 特定于数组的属性 **maxSize** 指示数组中的有效元素数目。缺省值是没有限制; 元素数目仅受到目标环境的要求限制。

您不必在声明中指定元素数目, 但如果指定了元素数目, 则该数目指示元素的初始数目。还可以通过在声明中列示一系列数组常量来指定元素的初始数目, 只能对基本变量这样做, 不能对记录这样做。

以下示例显示声明动态数组的语法:

```
// An array of 5 elements or less
myDataItem01 CHAR(30)[] { maxSize=5 };

// An array of 6 elements or less,
// with 4 elements initially
myDataItem02 myDataItemPart[4] { maxSize=6 };

// An array that has no elements
// but whose maximum size is the largest possible
myRecord myRecordPart[];

// A 3-element array whose elements
// are assigned the values 1, 3, and 5
position int[] = [1,3,5];
```

可使用文字整数来初始化元素数目, 但变量和常量都无效。

在声明数组的数组时, 元素的初始数目在从最左边指定的维中和每个后续维中有效, 直到某个维缺少初始编号为止。下列声明有效:

```
// Valid, with maxsize giving the maximum
// for the first dimension
myInt01 INT[3] [];
myInt02 INT[4][2] [] {maxsize = 12};
myInt03 INT[7][3][1];

// In the next example, array constants indicate
// that the outer array initially has 3 elements.
// The first element of the outer array is
// an array of two elements (with values 1 and 2).
// The second element of the outer array is
// an array of three elements (with values 3, 4,5).
```

```
// The third element of the outer array is
// an array of two elements (with values 6 and 7).
myInt04 INT[] [] = [[1,2],[3,4,5],[6,7]];
```

例如，以下示例中的语法无效，原因是数组 `myInt04` 被声明为没有元素的数组，但其中每个元素被指定了 3 个元素：

```
// NOT valid
myInt04 INT[][3];
myInt05 INT[5][][2];
```

被指定为程序或函数参数的数组不能指定元素数目。

当代码引用数组或数组元素时，下列规则适用：

- 元素下标可以是解析为整数的任何数字表达式，但该表达式不能包括函数调用。
- 如果代码引用动态数组但未指定下标，则引用的是整个数组。

内存不足状态被视为灾难性错误，并将导致程序结束。

动态数组函数

对每个动态数组都提供了一组函数和只读变量。在以下示例中，数组被称为 *series*：

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

数组的名称可能包括一组方括号，每个方括号中包含一个整数。下面是一个示例：

```
series INT[] [];

// resizes the second element
// of series, which is an array of arrays
series[2].resize(100);
```

在下面几节中，用数组名替换 *arrayName*，并注意，可以通过包名和 / 或库名来对名称进行限定。

appendAll():

```
arrayName.appendAll(appendArray Array in)
```

此函数执行下列操作：

- 通过添加由 *appendArray* 引用的数组副本，对 *arrayName* 引用的数组进行追加
- 根据添加的元素数目增大数组大小
- 对每个追加的元素指定适当的下标值

appendArray 的元素必须与 *arrayName* 的元素具有相同的类型。

appendElement()

```
arrayName.appendElement(content ArrayElement in)
```

此函数将一个元素放在指定数组的末尾并将大小增大 1。对于 *content*，可以替换为具有适当类型的变量；另外，可以指定一个文字，该文字将被赋予在操作期间创建的元素。此过程复制数据；如果指定变量，则该变量仍可用于比较或其它用途。

赋值指定了文字赋值规则。

getMaxSize()

`arrayName.getMaxSize ()` returns (INT)

此函数返回一个整数，它指示数组中允许的最大元素数目。

getSize()

`arrayName.getSize ()` returns (INT)

此函数返回一个整数，它指示数组中允许的元素数目。建议在处理动态数组时使用此函数而不是 `SysLib.size`。

另一函数提供的功能等同于 `arrayName.getSize` 的功能：

`SysLib.size ()` returns (INT)

但是，建议您在处理动态数组时使用 `arrayName.getSize ()`。

insertElement()

`arrayName.insertElement (content ArrayElement in, index INT in)`

此函数执行下列操作：

- 将一个元素放在目前位于数组中指定位置的元素之前
- 将数组大小增大 1
- 将位于所插入元素之后的每个元素的下标增大 1

`content` 是新内容（常量或变量，并具有适合于数组的类型），`index` 是指示新元素位置的整数文字或数字变量。

如果 `index` 比数组中的元素数目大 1，则此函数在数组末尾创建一个新元素并将数组大小增大 1。

removeAll()

`arrayName.removeAll ()`

此函数从内存中除去数组元素。该数组可以被使用，但它的大小为零。

removeElement()

`arrayName.removeElement(index INT in)`

此函数除去位于指定位置的元素，将数组大小减 1，并将位于所除去的元素后的每个元素的下标都减 1。

`index` 是一个整数文字或数字变量，它指示要除去的元素的位置。

resize()

`arrayName.resize(size INT in)`

此函数将数组的当前大小增加或缩小至 `size` 中指定的大小，该大小是一个整数文字、常量或变量。如果 `size` 的值大小数组允许的最大大小，运行单元将终止。

reSizeAll()

```
arrayName.reSizeAll(sizes INT[  
] in)
```

此函数增加或缩小多维数组的每个维。参数 *sizes* 是整数数组，每个连续元素指定连续维的大小。如果重新调整大小的维的数目大于 *arrayName* 中的维数，或者如果 *sizes* 中的元素的值大于 *arrayName* 的等同维中允许的最大大小，运行单元将终止。

setMaxSize()

```
arrayName.setMaxSize (size INT in)
```

该函数设置数组中允许的最大元素数目。如果将最大大小设置为小于数组当前大小的值，运行单元将终止。

setMaxSizes()

```
arrayName.setMaxSizes(sizes INT[  
] in)
```

此函数设置多维数组的每个维。参数 *sizes* 是整数数组，每个连续元素指定连续维的最大大小。如果指定的维的数目大于 *arrayName* 中的维数，或者如果 *sizes* 中的元素的值小于 *arrayName* 的等同维中的当前元素数目，运行单元将终止。

将动态数组用作自变量和参数

可以将动态数组作为自变量传递给 EGL 函数。必须将相关参数定义为与自变量具有相同类型的动态数组；对于数据项，类型必须包含相同的长度和小数位数（如果有的话）。

不能将动态数组作为自变量传递给另一个程序。

下面是将动态数组用作参数的函数的示例：

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

在运行时，参数的最大大小是对相应自变量声明的最大大小。函数或被调用程序可以更改数组的大小，此更改在调用代码中会生效。

SQL 处理和动态数组

EGL 允许使用动态数组来访问关系数据库的行。有关读取多行的详细信息，请参阅 *get*。有关添加多行的详细信息，请参阅 *add*。

结构字段数组

当您指定固定结构中的字段具有大于 1 的 *occurs* 值时，您就声明了结构字段数组，如下示例所示：

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

如果名为 *myRecord* 的固定记录基于该部件，则符号 *myRecord.mySi* 指的是包含三个元素的一维数组，其中每个元素都是一个字符。

结构字段数组的使用

在下列上下文中，可以引用结构字段的整个数组（例如 *myRecord.mySi*）：

- 作为 *in* 运算符使用的第二个操作数。此运算符测试数组是否包含给定的值。
- 作为函数 **sysLib.size** 中的参数。该函数返回结构字段的 *occurs* 值。

本身不是数组的数组元素是与任何其它项类似的字段，可以通过各种方式引用该字段；例如，在赋值语句中引用它，或者在函数调用中作为自变量来引用它。

元素下标可以是解析为整数的任何数字表达式，但该表达式不能包括函数调用。

一维结构字段数组

通过先指定数组名称并接着指定用方括号括起来的下标，可以引用一维数组（如 *myRecord.mySi*）的元素。下标是一个整数或解析为整数的字段；例如，可以使用 *myStruct.mySi[2]* 来引用示例数组中的第二个元素。下标的变化范围可以从 1 到结构字段的 *occurs* 值，如果下标超出该范围，则会发生运行时错误。

如果在需要字段的上下文中使用结构字段数组的名称，但未指定用方括号括起来的下标，则 EGL 假定您正在引用数组的第一个元素，但仅当处于 VisualAge Generator 兼容性方式时才如此。建议您显式地标识每个元素。如果未处于 VisualAge Generator 兼容性方式，则需要显式地标识每个元素。

下面的示例显示如何引用一维数组中的元素。在那些示例中，*valueOne* 解析为 1，*valueTwo* 解析为 2：

```
// these refer to the first of three elements:
myRecord.mySi[valueOne]

// not recommended; and valid
// only if VisualAge Generator
// compatibility is in effect:
myRecord.mySi

// this refers to the second element:
myRecord.mySi[valueTwo]
```

一维数组可以带有子结构，如以下示例所示：

```
record myRecord01Part
  10 name[3];
     20 firstOne CHAR(20);
     20 midOne CHAR(20);
     20 lastOne CHAR(20);
end
```

如果名为 *myRecord01* 的记录基于上一个部件，则符号 *myRecord01.name* 指的是包含三个元素的一维数组，其中每个元素都带有 60 个字符，*myRecord01* 的长度是 180。

可以在不引用子结构的情况下引用 *myRecord01.name* 中的每个元素；例如，*myRecord01.name[2]* 引用第二个元素。也可以引用元素中的子结构。例如，如果满足唯一性规则，则可以按照下列任何方式来引用第二个元素的后 20 个字符：

```
myRecord01.name.lastOne[2]
myRecord01.lastOne[2]
lastOne[2]
```

仅当可生成部件属性 **allowUnqualifiedItemReferences** 设置为 *yes* 时，最后两种方法才有效。

有关不同类型的引用的详细信息，请参阅[对变量和常量的引用](#)。

多维结构字段数组

如果 `occurs` 值大于 1 的结构项带有子结构，并且如果下级结构项也具有大于 1 的 `occurs` 值，则下级结构项声明带有附加维的数组。

让我们考虑另一个记录部件：

```
record myRecord02Part
  10 siTop[3];
  20 siNext CHAR(20)[2];
end
```

如果名为 `myRecord02` 的记录基于该部件，则一维数组 `myRecord02.siTop` 的每个元素本身也是一维数组。例如，可以使用 `myRecord02.siTop[2]` 来引用三个下级一维数组中的第二个一维数组。结构项 `siNext` 声明二维数组，您可以使用下列任一语法来引用该数组中的元素：

```
// row 1, column 2.
// the next syntax is strongly recommended
// because it works with dynamic arrays as well
myRecord02.siTop[1].siNext[2]

// the next syntax is supported for compatibility
// with VisualAge Generator
myRecord02.siTop.siNext[1,2]
```

要阐明引用了哪个内存区，应了解多维数组中的数据是如何存储的。在当前示例中，`myRecord02` 包含 120 个字节。引用的区域被划分为具有三个元素的一维数组，每个元素 40 个字节：

```
siTop[1]    siTop[2]    siTop[3]
```

一维数组的每个元素都进一步细分为一个包含两个元素的数组，其每个元素均为 20 个字节，并且位于同一个内存区中：

```
siNext[1,1] siNext[1,2] siNext[2,1] siNext[2,2] siNext[3,1] siNext[3,2]
```

二维数组是按行优先顺序来进行存储的。其含义是如果您以两个 `while` 循环来初始化数组，则可通过先处理一行中的列，再处理另一行中的列来提高性能：

```
// i, j, myTop0ccurs, and myNext0ccurs are data items;
// myRecord02 is a record; and
// sysLib.size() returns the occurs value of a structure item.
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
```

必须为多维数组的每个维指定值。例如，对于二维数组，引用 `myRecord02.siTop.siNext[1]` 是无效的。

以下是一个三维数组的声明示例：

```

record myRecord03Part
  10 siTop[3];
  20 siNext[2];
  30 siLast CHAR(20)[5];
end

```

如果名为 *myRecord03* 的记录基于该部件，并且如果满足唯一性规则，则可以按照下列任何方式来引用该数组中的最后一个元素：

```

// each level is shown, and a subscript
// is on each level, as is recommended.
myRecord03.siTop[3].siNext[2].siLast[5]

// each level shown, and subscripts are on lower levels
myRecord03.siTop.siNext[3,2].siLast[5]
myRecord03.siTop.siNext[3][2].siLast[5]

// each level is shown, and subscripts are on the lowest level
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]

// the container and the last level is shown, with subscripts
myRecord03.siLast[3,2,5]
myRecord03.siLast[3,2][5]
myRecord03.siLast[3][2,5]
myRecord03.siLast[3][2][5]

// only the last level is shown, with subscripts
siLast[3,2,5]
siLast[3,2][5]
siLast[3][2,5]
siLast[3][2][5]

```

如上一示例中所示，可通过添加用括号括起来的一组下标以任意方式引用多维数组的元素。在所有情况下，第一个下标表示第一个维，第二个下标表示第二个维，依此类推。每个下标的变化范围可以从 1 到相关结构项的 `occurs` 值，如果某个下标解析为该范围外的数字，则会发生运行时错误。

首先，考虑未涉及下标的情况：

- 可指定一个列表，该列表以变量名称开头，接着是一级一级的下级结构项的名称，并通过句点将各个名称隔开，如以下示例所示：

```
myRecord03.siTop.siNext.siLast
```

- 可指定变量的名称，后跟句点，然后是期望的最低级项的名称，如以下示例所示：

```
myRecord03.siLast
```

- 如果期望的最低级项在给定名称空间中是唯一的，则只能指定该项，如以下示例所示：

```
siLast
```

然后，考虑用于放置数组下标的规则：

- 可在其中一个元素有效的每一个级别指定下标，如以下示例所示：

```
myRecord03.siTop[3].siNext[2].siLast[5]
```

- 可在其中一个元素有效的任意级别指定一系列下标，如以下示例所示：

```
myRecord03.siTop.siNext[3,2].siLast[5]
```

- 可在任意级别指定一系列下标，这一级别位于其中一个元素有效的某个级别或是该级别的下级，如以下示例所示：

```
myRecord03.siTop.siNext.siLast[3,2,5]
```

- 如果在给定级别指定的下标过多，将发生错误，如以下示例所示：

```
// NOT valid
myRecord03.siTop[3,2,5].siNext.siLast
```

- 可用括号隔离下标，也可显示一系列下列，每个下标用逗号隔开；还可以同时使用这两种用法。下列示例有效：

```
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]
```

相关概念

第 400 页的『与 VisualAge Generator 的兼容性』

第 53 页的『引用 EGL 中的变量』

相关参考

第 511 页的『add』

第 340 页的『赋值』

第 451 页的『EGL 系统限制』

第 533 页的『get』

第 486 页的『in 运算符』

第 833 页的『size()』

字典

字典部件是一直可用的部件；不用定义它。基于字典部件的变量可能包括键及其相关值的集合，您可以在运行时添加和除去键和值条目。这些条目被视作记录中的字段。

以下是字典声明的一个示例：

```
row Dictionary
{
    ID          = 5,
    lastName   = "Twain",
    firstName  = "Mark",
};
```

在声明中包括条目时，每个键名都是一定符合 EGL 命名约定的 EGL 标识。在运行时添加条目时，可以更加灵活一些；可以指定字符串文字、常量或变量，在这种情况下，常量可以是 EGL 保留字或者可以包括在标识中无效的字符。有关详细信息，请参阅 *动态访问的括号语法*。

以下是一些赋值示例：

```
row.age = 30;
row["Credit"] = 700;
row["Initial rating"] = 500
```

如果尝试指定已经存在的键，则覆盖现有键与值条目。以下赋值有效并将“Twain”替换为“Clemens”：

```
row.lastname = "Clemens";
```

赋值也可以用于数据检索:

```
lastname String
age, credit, firstCredit int;

lastname = row.lastname;
age = row.age;
credit = row.credit;
credit = row["Credit"];
firstCredit = row["Initial rating"];
```

键与值条目中的值的类型为 ANY，这表示可将不同种类的信息放在单个字典中。每个值可以是下列任何一项:

- 预先声明的记录或其它变量
- 常量或文字

将变量放在字典中将给出变量的副本。考虑以下记录部件:

```
Record myRecordPart
  x int;
end
```

以下代码将类型为 myRecordPart 的变量放在字典中，然后更改原始变量中的值:

```
testValue int;

myRecord myRecordPart;

// sets a variable value and places
// a copy of the variable into the dictionary.
myRecord.x = 4;
row Dictionary
{
  theRecord myRecord;
}

// Places a new value in the original record.
myRecord.x = 700;

// Accesses the dictionary's copy of the record,
// assigning 4 to testValue.
testValue = row.theRecord.x;
```

将一个字典赋值给另一个字典会将目标内容替换为源内容，并且覆盖目标字典的属性值，如后所述。例如，以下代码中的条件语句求值为 true:

```
row Dictionary { age = 30 };

newRow Dictionary { };
newRow = row

// resolves to true
if (newRow.age == 30)
;
end
```

声明中的一组属性将影响处理字典的方式。一组特定于字典的函数向代码提供数据和服务。

字典属性

如以下示例中所示，每个属性与值条目在句法上相当于键与值条目，并且这些条目可以是任何顺序:

```

row Dictionary
{
  // properties
  caseSensitive = no,
  ordering = none,

  // fields
  ID = 5,
  lastName = "Twain",
  firstName = "Mark"
  age = 30;
};

```

代码既不能添加也不能检索属性或属性值。万一想要将属性名用作键，则在指定或引用键时将该变量名用作限定符，如以下示例中所示：

```

row Dictionary
{
  // properties
  caseSensitive = no,
  ordering = none,

  // fields
  row.caseSensitive = "yes"
  row.ordering = 50,
  age = 30
};

```

属性如下所示：

caseSensitive

指示存储相关值时使用的键的大小写是否会影响键或值的检索。选项如下所示：

No (缺省值)

键的访问不受键的大小写的影响，下列语句起到的作用是一样的：

```

age = row.age;
age = row.AGE;
age = row["aGe"];

```

Yes

执行下列语句会得到不同结果，即使 EGL 是根本不区分大小写的语言也是如此：

```

age = row.age;
age = row.AGE;
age = row["aGe"];

```

属性 **caseSensitive** 的值会影响后面一节中描述的若干函数的行为。

ordering

指示键与值条目用于检索的排序方式。如后面一节中所述，此属性的值会影响函数 **getKeys** 和 **getValues** 的行为。

选项如下所示：

None (缺省值)

代码不能依赖于键与值条目的顺序。

如果属性 **ordering** 的值为“None”，则键的顺序（在调用函数 **getKeys** 时）可能与值的顺序（在调用函数 **getValues** 时）不同。

ByInsertion

键 - 值对是以插入时的顺序提供的。声明中的所有条目被视为先以从左至右的顺序插入。

ByKey

键 - 值对是以键顺序提供的。

字典函数

要调用以下任何函数，用字典的名称来限定函数名，如以下示例中所示（此时字典被称为 *row*）：

```
if (row.containsKey(age))
;
end
```

containsKey()

dictionaryName.containsKey(key String in) returns (**Boolean**)

根据输入字符串 (*key*) 在字典中是否为键，此函数解析为 **true** 或 **false**。如果字典属性 **caseSensitive** 被设置为 **no**，则不考虑大小写；否则函数将搜索完全匹配，包括大小写匹配。

containsKey 仅在逻辑表达式中使用。

getKeys()

dictionaryName.getKeys () returns (**String[]**)

此函数返回一组字符串，每个字符串都是字典中的键。

如果字典属性 **caseSensitive** 被设置为 **no**，则每个返回键为小写；否则每个返回键为存储键时所使用的大小写。

如果字典属性 **ordering** 被设置为 **no**，则不能依赖于返回键的顺序；否则顺序为该属性的描述中指定的顺序。

getValues()

dictionaryName.getValues () returns (**ANY[]**)

此函数返回类型为 **ANY** 的一组值。每个值都与字典中的键相关联。

insertAll()

dictionaryName.insertAll(sourceDictionary Dictionary in)

此函数的行为方式就像一系列赋值语句将源字典 (*sourceDictionary*) 中的键与值条目复制至目标，目标也是一个字典，其名称限定函数名。

如果键在源中而不在目标中，则键与值条目将复制至目标。如果键既在源中又在目标中，则源条目的值将覆盖目标中该条目的值。在确定目标中的键是否与源中的键相匹配时，会受到每个字典中的属性 **caseSensitive** 的影响。

此函数不同于将一个字典赋值给另一个字典，原因是函数 **insertAll** 会保留下列条目：

- 目标中的属性与值条目；以及

- 在目标中但不在源中的键与值条目。

removeElement()

`dictionaryName.removeElement(key String in)`

此函数除去其输入字符串 (*key*) 充当字典中的键的条目。如果字典属性 **caseSensitive** 被设置为 `no`，则不考虑大小写；否则函数将搜索完全匹配，包括大小写匹配。

removeAll()

`dictionaryName.removeAll()`

此函数除去字典中的所有键与值条目，但不会影响字典的属性。

size()

`dictionaryName.size()` returns (INT)

返回一个整数，它指示字典中的键与值条目的数目。

相关概念

第 16 页的『部件』

第 53 页的『引用 EGL 中的变量』

相关参考

第 454 页的『逻辑表达式』

ArrayDictionary

`arrayDictionary` 部件是一直可用的部件；不用定义它。基于 `arrayDictionary` 部件的变量允许您通过检索每个数组中编号相同的元素来访问一系列数组。以此方式检索的元素集合本身就是字典，每个原始数组名被视作与数组元素中包含的值配对的键。

`arrayDictionary` 在与控制台用户界面中描述的显示技术相关联时特别有用。

下面的图形演示 `arrayDictionary`，它的声明包括名为 `ID`、`lastname`、`firstname` 和 `age` 的数组。椭圆围住的字典包括下列键与值条目：

```
ID = 5,  
lastName = "Twain",  
firstName = "Mark",  
age = 30
```

ID	LastName	FirstName	Age
5	Twain	Mark	30

您所关心的数组是字典数组，每个字典在演示时是从上至下列示而不是并排列示的。`arrayDictionary` 的声明需要数组的初始列表，但是，它们在演示时是并排列示的。

以下代码显示数组列表的声明，后跟使用这些数组的 `arrayDictionary` 的声明：

```

ID          INT[4];
lastName   STRING[4];
firstName  STRING[4];
age        INT[4];

myRows ArrayDictionary
{
    col1 = ID,
    col2 = lastName,
    col3 = firstName,
    col4 = age
};

```

要检索值，代码使用先分离出特定字典然后分离出该字典中的特定字段（键与值条目）的语法。不能使用 `arrayDictionary` 语法来更新值或更改 `arrayDictionary` 本身的任何特征。

首先，声明字典并将 `arrayDictionary` 行赋值给该字典，如以下示例中所示：

```
row Dictionary = myRows[2];
```

然后，声明相应类型的变量并将一个元素赋值给该变量，如以下示例中所示：

```
cell INT = row["ID"];

cell INT = row.ID;
```

另一语法只用一个步骤来检索值，如以下任一示例中所示：

```
cell int = myRows[2]["ID"];

cell int = myRows[2].ID;
```

相关概念

第 163 页的『控制台用户界面』

第 75 页的『字典』

第 53 页的『引用 EGL 中的变量』

EGL 语句

每个 EGL 函数可以不包含 EGL 语句也可以由多个下列类型的 EGL 语句组成：

- 变量声明或常量声明提供了对指定的内存区的访问。变量的值在运行时可以更改；常量的值不能更改。这两种类型的声明都可以位于函数中的除块以外的位置中，如后文所述。

- 函数调用将处理定向到一个函数中，如以下示例所示：

```
myFunction(myInput);
```

递归调用是有效的。

- 赋值语句可以将下列任何值复制到变量中：

- 常量或变量中的数据
- 文字
- 从函数调用返回的值
- 算术计算结果
- 字符串并置结果

以下是赋值语句的一些示例：

```
myItem = 15;
myItem = readFile(myKeyValue);
myItem = bigValue - 32;
record1.message = "Operation " + "successful!";
```

- 关键字语句提供了其它功能，如文件访问。这些语句中的每一个语句都是根据语句开头的关键字命名的；例如：

```
add record1;    // an add statement
return (0);    // a return statement
```

- 空语句是一个不起任何作用的分号，但它作为占位符可能很有用，如以下示例所示：

```
if (myItem == 5)
;    // a null statement
else
myFunction(myItem);
end
```

非空 EGL 语句具有下列特征：

- 语句可以引用指定的内存区，这些内存区具有下列类型：
 - 表单
 - PageHandler
 - 记录
 - 数据表
 - 项（包括数据项以及记录、表单和表中的结构项的类别）
 - 数组（基于具有大于 1 的 occurs 值的结构项的内存区）
- 语句可以包含下列类型的表达式：
 - 日期时间表达式解析为日期、整数、时间间隔或时间戳记
 - 逻辑表达式解析为 true 或 false
 - 数字表达式解析为数字，它可能带有符号并包含小数点
 - 字符串表达式解析为一系列字符，它可能包含单字节字符、双字节字符或这两者的组合

- 语句以分号或块结束，块是作为一个单元的零个或多个下级语句的序列。包含块的语句以 `end` 定界符终止，如以下示例所示：

```

if (record2.status= "Y")
    record1.total = record1.total + 1;
    record1.message = "Operation successful!";
else
    record1.message = "Operation failed!";
end

```

`end` 定界符后面的分号不是错误，而是被视为空语句。

语句和整个 EGL 中的名称都是不区分大小写的；例如，`record1` 与 `RECORD1` 完全相同，`add` 与 `ADD` 指的是同一个关键字。

注：在 Page Designer 中使用源代码选项卡时，可手工将 JSP 文件（具体地说，JavaServer Faces 文件）中的组件与 PageHandler 中的数据区绑定。尽管 EGL 是不区分大小写的，但 JSP 文件中引用的 EGL 变量名的大小写必须与 EGL 变量声明的大小写相同；如果未能保持完全匹配，将发生 JavaServer Faces 错误。建议在将该变量与 JSP 字段绑定后不要更改 EGL 变量的大小写。

系统字是一组提供特殊功能的字：

- 系统函数运行代码并有可能返回值；例如：
 - `sysLib.minimum(arg1, arg2)` 返回两个数中的较小者
 - `strLib.strLen(arg1)` 返回字符串的长度

仅当程序包含同名的函数时，限定符 (`mathLibstrLib` 或 `sysLib`) 才是必需的。

- 系统变量在不调用函数的情况下提供一个值；例如：
 - 在程序访问文件之后以及在其它情况下，`sysVar.errorCode` 包含状态码
 - 在程序访问关系数据库之后，`sysVar.sqlcode` 包含状态码

仅当程序包含同名的变量时，限定符 `sysVar` 才是必需的。

函数中的一行可以包含多个语句。但是，因为只能在一行的第一个语句上使用 EGL 调试器来设置断点，所以建议您不要在一行上包括多个语句。

另请参阅注释。

相关概念

- 第 13 页的『EGL 项目、包和文件』
- 第 130 页的『函数部件』
- 第 16 页的『部件』

相关参考

- 第 511 页的『add』
- 第 340 页的『赋值』
- 第 513 页的『call』
- 第 515 页的『case』
- 第 517 页的『close』
- 第 399 页的『注释』
- 第 430 页的『数据初始化』
- 第 520 页的『delete』

第 444 页的『EGL 保留字』
 第 522 页的『execute』
 第 473 页的『函数调用』
 第 533 页的『get』
 第 544 页的『get next』
 第 549 页的『get previous』
 第 555 页的『if, else』
 『按字母顺序排列的关键字』
 第 454 页的『逻辑表达式』
 第 461 页的『数字表达式』
 第 561 页的『open』
 第 574 页的『prepare』
 第 576 页的『replace』
 第 579 页的『set』
 第 462 页的『文本表达式』
 第 860 页的『terminalID』
 第 591 页的『while』

按字母顺序排列的关键字

关键字	用途
第 511 页的『add』	将记录放到文件、消息队列或数据库中；或者将一组记录放到数据库中。
第 513 页的『call』	将控制权转移至另一个程序并传递（可选）一系列的值。当被调用程序结束时，控制权又返回至调用程序。如果被调用程序更改了任何通过变量传递的数据，则可供调用程序使用的存储区域也会更改。
第 515 页的『case』	标记多组语句的开始，在该位置，最多只运行那些语句组中的其中一组。 case 语句等同于在每个 case 子句末尾都有一个 break 的 C 或 Java <i>switch</i> 语句。
第 517 页的『close』	断开与打印机的连接；或者关闭与给定记录相关联的文件或消息队列；或者，对于 SQL 记录，关闭由 EGL open 或 get 语句打开的游标。
第 519 页的『continue』	在文本应用程序中显示文本表单。
第 519 页的『converse』	在文本应用程序中显示文本表单。
第 520 页的『delete』	从文件中除去一条记录或者从数据库中除去一行。
第 522 页的『display』	将一个文本表单添加到运行时缓冲区中，但不将数据显示到屏幕上。
第 522 页的『execute』	允许您编写一个或多个 SQL 语句；特别是，允许您编写 SQL 数据定义语句（例如，CREATE TABLE 类型的数据定义语句）和数据处理语句（例如，INSERT 或 UPDATE 类型的数据处理语句）。
第 526 页的『exit』	离开指定的块，缺省情况下，该块是直接包含 exit 语句的块。
第 528 页的『for』	开始一个语句块，测试求值为 true 时该语句块会循环运行多次。
第 530 页的『forEach』	标记在循环中运行的一组语句的开始。仅当指定的结果集可用时才会发生第一次迭代，并且该迭代会持续（在大多数情况下）至处理结果集的最后一行。

关键字	用途
第 531 页的 『forward』	显示带有变量信息的 Web 页面。此语句是从 PageHandler 中调用的。
第 532 页的 『freeSQL』	释放与动态预编译 SQL 语句相关联的所有资源，关闭与该 SQL 语句相关联的任何打开游标。
第 533 页的 『get』	检索单个文件记录或数据库行，并提供了一个选项以允许以后在代码中替换或删除所存储的数据。另外，此语句允许检索一组数据库行并将每个后续行放到动态数组中的下一个 SQL 记录中。 get 语句有时被标识为 get by key value ，它与 get by position 语句（如 get next ）是不同的。
第 538 页的 『get absolute』	读取关系数据库结果集中由 open 语句选择的用数字指定的行。
第 540 页的 『get current』	读取关系数据库结果集中由 open 语句选择的游标已经定位的行。
第 541 页的 『get first』	读取数据库结果集中由 open 语句选择的第一行。
第 543 页的 『get last』	读取数据库结果集中由 open 语句选择的最后一行。
第 544 页的 『get next』	从文件或消息队列中读取下一条记录，或者从数据库结果集中读取下一行。
第 549 页的 『get previous』	读取文件中与指定 EGL 带索引记录相关联的上一条记录；或者读取关系数据库结果集中由 open 语句选择的上一行。
第 552 页的 『get relative』	读取数据库结果集中由 open 语句选择的用数字指定的行。该行是根据结果集中的游标位置指定的。
第 554 页的 『goTo』	致使处理在指定的标号处继续，该标号必须与该语句位于同一个函数中并且位于块外部。
第 555 页的 『if, else』	标记仅当逻辑表达式解析为 true 时才会运行的一组语句（如果有的话）的开始。可选关键字 else 标记备用语句组（如果有的话）的开始，仅当逻辑表达式解析为 false 时才会运行该组语句。保留字 end 标记 if 语句的结束。
第 556 页的 『move』	复制数据，逐个字节复制或者按名称复制。按名称复制会将数据从一个结构中的命名项复制至另一个结构中的同名项。
第 561 页的 『open』	从关系数据库中选择一组行，以供 get by position 语句（如 get next ）以后检索。 open 语句可以对游标或对被调用过程执行操作。
第 574 页的 『prepare』	指定一个 SQL PREPARE 语句，后者包含（可选）只有在运行时才知道的详细信息。通过运行 EGL execute 语句或者（如果 SQL 语句返回结果集的话）通过运行 EGL open 或 get 语句来运行预编译 SQL 语句。
第 576 页的 『print』	将打印表单添加至运行时缓冲区。
第 576 页的 『replace』	将已更改的记录放到文件或数据库中。
第 579 页的 『return』	从函数中退出并返回（可选）一个值给调用函数。
第 579 页的 『set』	会对记录、文本表单和项产生各种作用。

关键字	用途
第 588 页的『show』	从主程序中显示文本表单以及使用 <code>display</code> 语句缓存的任何其它表单；结束当前程序并（可选）将用户提供的输入数据以及当前程序提供的状态数据转发至复杂处理用户输入的程序。
第 589 页的『transfer』	将控制权从一个主程序转移到另一个主程序，结束转移程序，并传递（可选）一条记录，该记录的数据被接受到接收程序的输入记录。不能在被调用程序中使用 transfer 语句。
第 590 页的『try』	指示当输入 / 输出 (I/O) 语句、系统函数调用或 call 语句导致错误并且位于该 try 语句之内时，程序继续运行。如果发生异常，则在 onException 块（如果有的话）中的第 1 个语句处继续处理，或者在 try 语句后面的第 1 个语句处继续处理。但是，仅当系统变量 VGVar.handleHardIOErrors 设置为 1 时才处理硬 I/O 错误；否则，程序显示消息（如果有可能的话）并结束。
第 591 页的『while』	标记在循环中运行的一组语句的开始。仅当逻辑表达式解析为 true 时，才发生第一次运行，且每一后续迭代都取决于相同的测试。保留字 end 标记 while 语句的结束。

相关参考

第 80 页的『EGL 语句』

在程序之间转移控制权

EGL 提供了几种方法来将控制权从一个程序切换到另一个程序：

- **call** 语句将控制权给予另一个程序并可以选择传递一系列值。当被调用程序结束时，控制权又返回至调用程序。如果被调用程序更改了任何作为变量传递的数据，则在调用程序中更改变量的内容。

虽然可能会发生服务器端自动落实，但是 **call** 语句不落实数据库或其它可恢复的资源。

可以通过设置链接选项部件的 `callLink` 元素来指定 **call** 语句的特征。有关详细信息，请参阅 `call` 和 `callLink` 元素。有关服务器端自动落实的详细信息，请参阅 `callLink` 元素中的 `luwControl`。

- **transfer** 语句将控制权从一个主程序转移到另一个主程序，结束转移程序，并传递（可选）一条记录，该记录的数据被接受到接收程序的输入记录中。不能在被调用程序中使用 **transfer** 语句。

程序可以通过 `transfer to a transaction` 格式的语句或 `transfer to a program` 格式的语句来转移控制权：

- `transfer to a transaction` 执行下列操作：
 - 在作为 Java 主文本程序或主批处理程序运行的程序中，行为取决于构建描述符选项 `synchOnTrxTransfer` 的设置：
 - 如果 `synchOnTrxTransfer` 的值为 **YES**，则 **transfer** 语句将落实可恢复的资源、关闭文件、关闭游标并在同一个运行单元中启动程序。
 - 如果 `synchOnTrxTransfer` 的值为 **NO**（缺省值），则 **transfer** 语句也在同一个运行单元中启动程序，但不关闭或落实可供被调用程序使用的资源。
- `transfer to a program` 不落实或回滚可恢复的资源，但关闭文件、释放锁并在同一个运行单元中启动程序。

链接选项部件不影响任何一种类型的转移的特征。

在 PageHandler 中，转移是无效的。

有关详细信息，请参阅 *transfer*。

- 系统函数 **sysLib.startTransaction** 以异步方式启动运行单元。此操作不结束转移程序也不影响数据库、文件和转移程序中的锁。您可以选择将数据传递到输入记录中，输入记录是接收程序中的一个区域。

如果程序调用 **sysLib.startTransaction**，则必须使用链接选项部件的 **asynchLink** 元素来生成程序。有关详细信息，请参阅 *sysLib.startTransaction* 和 *asynchLink* 元素。

- EGL **show** 语句结束文本应用程序中的当前主程序并通过表单向用户显示数据。（可选）用户提交表单后，**show** 语句将控制权转发至第二个主程序，该主程序接收从用户那里接收到的数据以及从起始程序按原样传递的数据。

show 语句受链接选项部件的 **transferLink** 元素设置的影响。

有关详细信息，请参阅 *show*。

- 最后，**forward** 语句是从程序或 PageHandler 中调用的。此语句执行下列操作：
 1. 落实可恢复的资源、关闭文件并释放锁
 2. 转发控制权
 3. 结束代码

在这种情况下，目标是另一个程序或 Web 页面。有关详细信息，请参阅 *forward*。

相关参考

- 第 343 页的『*asynchLink* 元素』
- 第 513 页的『*call*』
- 第 370 页的『*callLink* 元素』
- 第 531 页的『*forward*』
- 第 377 页的『*callLink* 元素中的 *luwControl*』
- 第 588 页的『*show*』
- 第 835 页的『*startTransaction()*』
- 第 589 页的『*transfer*』

异常处理

当 EGL 生成的程序执行下列操作时，可能会发生错误：

- 访问文件、队列或数据库
- 调用另一个程序
- 调用函数
- 执行赋值、比较或计算

try 块

EGL *try* 块可以不包含 EGL 语句也可以是包含多个 EGL 语句的序列，这些 EGL 语句位于定界符 **try** 与 **end** 之间。下面是一个示例：

```

if (userRequest = "A")
  try
    add record1;
  onException
    myErrorHandler(12);
  end
end

```

通常，try 块允许程序在发生错误的情况下继续进行处理。

try 块可以包含 onException 子句，如上所示。如果 try 块中的其中一个先前语句失败，则将调用该子句；但是，在不存在 onException 子句的情况下，try 块中的错误将导致调用紧跟在 try 块后的第一个语句。

EGL 系统异常

EGL 提供一系列系统异常以指示运行时问题的具体特性。其中每个异常都是一个字典，您可以从中检索信息，但检索总是通过系统变量 **SysLib.currentException**（也是一个字典）进行的，它允许您访问运行单元中最近抛出的异常。

任何异常中都有一个字段 **code**，它是标识该异常的字符串。可以通过以下逻辑测试该字段来确定当前异常：

```

if (userRequest = "A")
  try
    add record1;
  onException
    case (SysLib.currentException.code)
      when (FileIOException)
        myErrorHandler(12);
      otherwise
        myErrorHandler(15);
    end
  end
end

```

在此情况下，FileIOException 是常量，它相当于字符串值“com.ibm.egl.FileIOException”。EGL 异常常量总是等同于以“com.ibm.egl”开头的字符串中的最后一个限定符。

强烈建议只访问 onException 块中的异常字段。如果代码在没有发生任何异常的情况下访问 **SysLib.currentException**，运行单元将终止。

下一个示例访问异常 SQLException 中的字段 sqlcode:

```

if (userRequest = "A")
  try
    add record01;
  onException
    case (SysLib.currentException.code)
      when ("com.ibm.egl.SQLException")
        if (SysLib.currentException.sqlcode == -270)
          myErrorHandler(16);
        else
          myErrorHandler(20);
        end
      otherwise
        myErrorHandler(15);
    end
  end
end

```

有关系统异常的详细信息，请参阅 *EGL 系统异常*。

try 块的限制

必须符合有关 try 块的先前详细信息。首先，try 块只影响下列类型的 EGL 语句的错误处理：

- I/O 语句
- 系统函数
- call 语句

是否存在 try 块并不影响数字溢出处理。有关这些类型的错误的详细信息，请参阅 *VGVar.handleOverflow*。

其次，try 块对从该 try 块中调用的用户函数（或程序）内的错误不起作用。在下面的示例中，如果函数 myABC 中的语句失败，则除非函数 myABC 本身处理该错误，否则程序将立即结束并发出错误消息：

```
if (userRequest = "B")
  try
    myVariable = myABC();
  onException
    myErrorHandler(12);
  end
end
```

再次，在下列情况下，程序将立即结束并发出错误消息：

- 由某一 try 块专门处理的一类错误发生在该 try 块外部；或者
- 存在下列其中一种情况，甚至是在 try 块中：
 - 在调用或返回用户编写的函数时失败；或者
 - 将非数字字符赋给数字变量；或者
 - 当文件 I/O 语句由于硬错误而结束时，系统变量 **VGVar.handleHardIOErrors** 被设置为 0 而不是 1（如后文所述）。

下列情况也很重要：

- 如果某个值除以零，则 Java 程序将这种情况作为数字溢出来处理
- 如果将非数字字符赋值给数字变量，则 Java 程序将结束

注：为了支持迁移用 VisualAge Generator 和 EGL 5.0 编写的程序，变量 *VGVar.handleSysLibraryErrors*（以前称为 *ezereply*）允许处理一些在 try 块之外发生的错误。避免使用该变量，仅当您以 VisualAge Generator 兼容性方式工作时该变量才可用。

与错误相关的系统变量

EGL 提供了与错误相关的系统变量，在 try 块中设置这些变量以作为对成功事件的响应或而非终止错误的响应。在 try 块中以及在跟在 try 块后面运行的代码中可以使用那些变量的值，在大多数情况下，在转换（如果有的话）之后会恢复那些值。

当语句在 try 块外部运行时，EGL 运行时不更改任何与错误相关的变量的值。但是，程序可以在 try 块外部对与错误相关的变量赋值。

将在各种情况下对系统变量 **sysVar.exceptionCode** 赋值，在所有那些情况下，还将设置一个或多个其它变量，这取决于程序与运行时环境进行的交互的性质：

- 在 `try` 块中运行任何下列类型的语句之后，将对系统变量 **sysVar.exceptionCode** 和 **sysVar.errorCode** 赋值：
 - `call` 语句
 - 对索引文件、MQ 文件、相对文件或串行文件操作的 I/O 语句
 - 对几乎任何系统函数的调用
- 在 `try` 块中的 I/O 语句对 MQ 记录执行操作之后，将对系统变量 **sysVar.exceptionCode**、**sysVar.errorCode**、**VGVar.mqConditionCode** 和 **sysVar.mqReasonCode** 赋值
- 在从 `try` 块中的语句中访问关系数据库之后，将对系统变量 **sysVar.exceptionCode** 赋值。还将对 SQL 通信区 (SQLCA) 中的变量赋值；有关详细信息，请参阅 *sysVar.sqlca*。

如果在 `try` 块中发生非终止错误，则 **sysVar.exceptionCode** 的值等同于当在 `try` 块外部发生错误时将向用户显示的 EGL 错误消息的数字组件。但是，特定于情况的变量（如 **sysVar.errorCode** 和 **VGVar.mqConditionCode**）的值是由运行时系统提供的。未发生错误时，**sysVar.exceptionCode** 的值与至少一个特定于情况的变量相同：8 个零的字符串。

发生非终止数字溢出时，将会把错误代码赋给 **sysVar.exceptionCode** 和 **sysVar.errorCode**，如 *VGVar.handleOverflow* 所述；但是，成功的算术计算不影响任何与错误相关的系统变量。

与错误相关的系统变量也不受除系统函数外的函数调用的影响，并且，下列各项中的错误不影响 **sysVar.errorCode**（这是一个受大多数系统函数影响的变量）：

- **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**

当将错误值赋给 **sysVar.exceptionCode** 时，将把相关 EGL 错误消息的文本赋给系统变量 **sysVar.exceptionMsg**，并将错误消息中的字节数（不包括结尾空格和 NULL）

赋给系统变量 **sysVar.exceptionMsgCount**。在将 8 个零的字符串赋给 **sysVar.exceptionCode** 时，将把空白赋给 **sysVar.exceptionMsg**，并将把 **sysVar.exceptionMsgCount** 设置为 0。

I/O 语句

对于 I/O 语句，可以出现硬错误也可以出现软错误：

- 软错误是下列任何一项：
 - 在对 SQL 数据库表执行 I/O 操作期间找不到记录
 - 在对索引文件、相对文件或串行文件执行 I/O 操作时发生下列其中一个问题：
 - 记录重复（当外部数据存储允许插入重复记录时）
 - 找不到任何记录
 - 文件结束
- 硬错误就是任何其它问题；例如：
 - 记录重复（当外部数据存储禁止插入重复记录时）
 - 找不到文件
 - 通信链路在数据集远程访问期间不可用

如果导致软错误的语句在 `try` 块中，则下列陈述适用：

- 在缺省情况下，EGL 继续运行并且不会将控制权转交给 `onException` 块
- 如果希望将控制权转交给 `OnException` 块，则在程序、`pageHandler` 或库中将属性 **throwNrfEofExceptions** 设置为 `yes`

如果在 `try` 块中发生硬 I/O 错误，则后果取决于与错误相关的系统变量的值：

- 在访问文件、关系数据库或 MQSeries 消息队列期间，下列规则适用：
 - 如果 **VGVar.handleHardIOErrors** 设置为 1，则程序继续运行
 - 如果 **VGVar.handleHardIOErrors** 设置为 0，则程序显示错误消息（如果有可能的话）并结束

该变量的缺省设置取决于属性 **handleHardIOErrors** 的值，该属性在可生成逻辑部件（如程序、库和 `pageHandler`）中可用。该属性的缺省值为 `yes`，它将变量 **VGVar.handleHardIOErrors** 的初始值设置为 1。

如果在 `try` 块外部发生硬 I/O 错误或软 I/O 错误，则生成的程序显示错误消息（如果有可能的话）并结束。

如果正在直接访问 DB2（而不是通过 JDBC 进行访问），则硬错误的 `sqlcode` 是 304、802 或小于 0。

错误标识

通过在 `try` 块内部或外部包括 `case` 或 `if` 语句，可以确定在 `try` 块中发生的错误的类型，并且可以在该语句中测试各种系统变量的值。但是，如果您正在响应 I/O 错误并且如果语句使用 EGL 记录，则建议您使用基本逻辑表达式。有两种表达式格式：

```
recordName is IOerrorValue
```

```
recordName not IOerrorValue
```

recordName

在 I/O 操作中使用的记录的名称

IErrorValue

多个 I/O 错误值的其中一个，这些错误值在不同数据库管理系统之间是固定不变的

如果不将逻辑表达式与 I/O 错误值配合使用而后更改数据库管理系统，则可能需要修改并重新生成程序。尤其是，建议您使用 I/O 错误值来测试错误，而不是使用 **sysVar.sqlcode** 或 **sysVar.sqlState** 的值。那些值取决于底层数据库实现。

相关概念

第 400 页的『与 VisualAge Generator 的兼容性』

第 75 页的『字典』

相关参考

第 879 页的『EGL Java 运行时错误代码』

第 80 页的『EGL 语句』

第 490 页的『I/O 错误值』

第 454 页的『逻辑表达式』

第 853 页的『errorCode』

第 854 页的『overflowIndicator』

第 857 页的『sqlca』

第 858 页的『sqlcode』

第 858 页的『sqlState』

第 869 页的『handleSysLibraryErrors』

第 867 页的『handleHardIOErrors』

第 868 页的『handleOverflow』

第 869 页的『mqConditionCode』

将 EGL 代码迁移至 EGL 6.0 iFix

EGL V6.0 迁移工具转换 V5.1.2 和 V6.0 中的 EGL 源代码以符合 EGL V6.0 iFix。可对整个项目、单个文件或选择的一组文件使用此工具。对包或文件夹运行该工具会转换该包或文件夹中的所有 EGL 源文件。有关迁移工具更改的代码的更多信息，请参阅 *EGL 至 EGL 迁移*。

注： 不要对已经更新至 EGL V6.0 iFix 的代码使用迁移工具。这样做可能会导致代码中出现错误。

要将 EGL 代码迁移至 EGL V6.0 iFix，执行下列操作：

1. 在工作台中，单击 **窗口 > 首选项**。
2. 在“首选项”窗口的左边，展开 **工作台**，然后单击 **功能**。
3. 从功能列表中选择 **EGL Developer**。
4. 选择名为 **EGL V6.0 迁移** 的功能的复选框。
5. 单击 **确定**。
6. 再次单击 **窗口 > 首选项**。
7. 在“首选项”窗口的左边，展开 **EGL**，然后单击 **EGL V6.0 迁移首选项**。
8. 为 EGL V6.0 迁移工具设置首选项。有关此窗口中的首选项的更多信息，请参阅设置 *EGL 至 EGL 迁移首选项*。
9. 在“项目资源管理器”视图或“导航器”视图中，选择想要迁移的 EGL 项目、包、文件夹或文件。可选择迁移任何数目的 EGL 资源。要一次选择多个资源，在单击资源时按住 **CTRL**。
10. 右键单击所选资源并从弹出菜单中单击 **EGL V6.0 迁移 > 迁移**。
11. 检查代码以了解是否存在不符合 EGL V6.0 iFix 的情况。

迁移工具转换所选 EGL 源文件以符合 EGL V6.0 iFix。要查看工具对源代码所作的更改，执行下列操作：

1. 在“项目资源管理器”视图或“导航器”视图中，右键单击已迁移的 EGL 源文件并从弹出菜单中单击 **比较对象 > 本地历史记录**。
2. 检查工作空间中的文件与上一版本之间的差别。
3. 在检查完更改后，单击 **确定**。

相关概念

第 94 页的『EGL 至 EGL 迁移』

第 102 页的『设置 EGL 至 EGL 迁移首选项』

相关任务

第 113 页的『启用 EGL 功能』

EGL 至 EGL 迁移

EGL V6.0 迁移工具转换 V5.1.2 和 V6.0 中的 EGL 源代码以符合 EGL V6.0 iFix。可对整个项目、单个文件或选择的一组文件使用此工具。对包或文件夹运行该工具会转换该包或文件夹中的所有 EGL 源文件。有关如何使用迁移工具的指示信息，请参阅将 *EGL 代码迁移至 EGL 6.0 iFix*。

迁移工具可将注释添加至它更改的每个文件，还可以将注释添加至项目的日志文件。要更改这些选项，请参阅 *EGL 至 EGL 迁移首选项*。

迁移工具按下列方式更改 EGL 代码以符合 EGL V6.0 iFix:

- 迁移工具更改指定属性的方式。有关对属性的更改的信息，请参阅 *EGL 至 EGL 迁移期间对属性的更改*。
- 迁移工具搜索与保留字有冲突的变量和部件名。迁移工具通过按 EGL 至 EGL 迁移首选项中定义的那样添加前缀或后缀来更改这些变量和部件名。在缺省情况下，该工具将后缀 `_EGL` 添加至现在是保留字的任何名称。迁移工具不会重命名 CALL 语句的对象，也不会更新 EGL 构建部件文件中的引用。请参阅 *EGL 保留字*。以下是使用迁移工具之前和之后的代码示例。

迁移之前:

```
Library Handler
  boolean Bin(4);
End
```

迁移之后:

```
Library Handler_EGL
  boolean_EGL Bin(4);
End
```

- 在用作比较运算符时，迁移工具将一个等号 (=) 替换为两个等号 (==)。在用作赋值运算符时，它不会更改单个等号。

迁移之前:

```
Function test(param int)
  a int;
  If(param = 3)
    a = 0;
  End
End
```

迁移之后:

```
Function test(param int)
  a int;
  If(param == 3)
    a = 0;
  End
End
```

- 迁移工具将级别号添加至没有级别号的记录。

迁移之前:

```
Record MyRecord
  item1 int;
  item2 int;
End
```

迁移之后:

```
Record MyRecord
  10 item1 int;
  10 item2 int;
End
```

- 迁移工具更改常量的声明语法。

迁移之前:

```
intConst 3;
```

迁移之后:

```
const intConst int = 3;
```

- 迁移工具更改已移至不同库或已经重命名的变量和函数名。此更改会影响 SysLib 和 SysVar 库中的变量和函数。

迁移之前:

```
SysLib.java();
clearRequestAttr();
```

迁移之后:

```
JavaLib.invoke();
J2EELib.clearRequestAttr();
```

下面是 SysLib 和 SysVar 库中的已更改变量和函数名的列表:

表 1. SysLib 和 SysVar 库中的已更改变量和函数名

迁移之前	迁移之后
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
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

表 1. SysLib 和 SysVar 库中的已更改变量和函数名 (续)

迁移之前	迁移之后
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.systemGregorianDateFormat	VGVar.systemGregorianDateFormat
SysVar.systemJulianDateFormat	VGVar.systemJulianDateFormat
SysVar.currentDate	VGVar.currentGregorianDate
SysVar.currentFormattedDate	VGVar.currentFormattedGregorianDate
SysVar.currentFormattedJulianDate	VGVar.currentFormattedJulianDate
SysVar.currentFormattedTime	VGVar.currentFormattedTime
SysVar.currentJulianDate	VGVar.currentJulianDate
SysVar.currentShortDate	VGVar.currentShortGregorianDate
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
SysVar.commitOnConverse	ConverseVar.commitOnConverse
SysVar.eventKey	ConverseVar.eventKey
SysVar.printerAssociation	ConverseVar.printerAssociation
SysVar.segmentedMode	ConverseVar.segmentedMode
SysVar.validationMsgNum	ConverseVar.validationMsgNum

- 迁移工具更改指定日期、时间和时间戳记的方式。下面是一些示例:

表 2. 对日期、时间和时间戳记的更改

迁移之前	迁移之后
dateFormat = "yy/mm/dd"	dateFormat = "yy/MM/dd"
dateFormat = "YYYY/MM/DD"	dateFormat = "yyyy/MM/dd"
dateFormat = "YYYY/DDD"	dateFormat = "yyyy/DDD"
timeFormat = "hh:mm:ss"	timeFormat = "HH:mm:ss"

- 对于未指定属性的所有已迁移库、程序和 PageHandler，迁移工具将属性 HandleHardIOErrors 设置为 no。

相关任务

第 93 页的『将 EGL 代码迁移至 EGL 6.0 iFix』

相关概念

第 102 页的『设置 EGL 至 EGL 迁移首选项』

『EGL 至 EGL 迁移期间的属性更改』

相关参考

第 444 页的『EGL 保留字』

EGL 至 EGL 迁移期间的属性更改

迁移工具对指定属性的方式作了显著的更改。以下是这些更改的总结:

- 迁移工具重命名其名称在 EGL V6.0 iFix 中已更改的属性。以下是已重命名属性的列表:

表 3. 重命名属性

迁移之前	迁移之后
action	actionFunction
boolean	isBoolean
getOptions	getOptionsRecord
msgDescriptor	msgDescriptorRecord
onPageLoad	onPageLoadFunction
openOptions	openOptionsRecord
putOptions	putOptionsRecord
queueDescriptor	queueDescriptorRecord
range	validValues
rangeMsgKey	validValuesMsgKey
selectFromList	selectFromListItem
sqlVar	sqlVariableLen
validator	validatorFunction
validatorMsgKey	validatorFunctionMsgKey
validatorTable	validatorDataTable
validatorTableMsgKey	validatorDataTableMsgKey

- 迁移工具为用作字符串文字的属性值加上双引号。

迁移之前:

```
{ alias = prog }
```

迁移之后:

```
{ alias = "prog" }
```

下列属性会受影响:

- alias
 - column
 - currency
 - displayName
 - fileName
 - fillCharacter
 - help
 - helpKey
 - inputRequiredMsgKey
 - minimumInputMsgKey
 - msgResource
 - msgTablePrefix
 - pattern
 - queueName
 - rangeMsgKey
 - tableNames
 - title
 - typeChkMsgKey
 - validatorMsgKey
 - validatorTableMsgKey
 - value
 - view
- 在将数组文字指定为属性值时，迁移工具将圆括号替换为方括号。
 - formSize
 - keyItems
 - outline
 - pageSize
 - position
 - range
 - screenSize
 - screenSizes
 - tableNames
 - tableNameVariables

- validationBypassFunctions
- validationBypassKeys
- 对于采用数组文字的属性，迁移工具会将单元素数组文字用方括号括起来，以指定只有一个元素的数组仍然是数组。迁移工具使用两对方括号来表示采用数组的数组的属性。

迁移之前:

```
{ keyItems = var, screenSize = (24, 80), range = (1, 9) }
```

迁移之后:

```
{ keyItems = ["var"], screenSize = [[24, 80]], range = [[1, 9]] }
```

- 在覆盖数组中的特定元素的属性时，迁移工具使用关键字 **this**（而不是变量名）。

迁移之前:

```
Form myForm type TextForm
  fieldArray char(10)[5] { fieldArray[1] {color = red } };
end
```

迁移之后:

```
Form myForm type TextForm
  fieldArray char(10)[5] { this[1] {color = red } };
end
```

- 迁移工具更改对部件、函数和字段的引用，在适当时加上引号和方括号。

迁移之前:

```
{ keyItems = (item1, item2) }
```

迁移之后:

```
{ keyItems = ["item1", "item2"] }
```

下列属性受到迁移工具的影响:

- action
- commandValueItem
- getOptions
- helpForm
- inputForm
- inputPageRecord
- inputRecord
- keyItem
- keyItems
- lengthItem
- msgDescriptorRecord
- msgField
- numElementsItem
- onPageLoadFunction
- openOptionsRecord

- putOptionsRecord
 - queueDescriptorRecord
 - redefines
 - selectFromListItem
 - tableNameVariables
 - validationBypassFunctions
 - validatorFunction
 - validatorDataTable
- 迁移工具将缺省值 `yes` 赋给已指定但未赋值的布尔属性。

迁移之前:

```
{ isReadOnly }
```

迁移之后:

```
{ isReadOnly = yes }
```

下列属性受到迁移工具的影响:

- 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
- 迁移工具将 **currency** 属性分割为两个属性: **currency** 和 **currencySymbol**。下表给出迁移工具如何更改 **currency** 属性的一些示例。

表 4. 对 **currency** 属性的更改

迁移之前	迁移之后
{ currency = yes }	{ currency = yes }
{ currency = no }	{ currency = no }
{ currency = "usd" }	{ currency = yes, currencySymbol = "usd" }

- 迁移工具将 **dateFormat** 和 **timeFormat** 的属性的值更改为区分大小写。有关更多信息, 请参阅日期、时间和时间戳记格式说明符。
- 如果在首选项菜单中选择了将限定符添加至枚举属性值复选框, 则迁移工具会将值类型添加至属性值。

迁移之前:

```
color = red
outline = box
```

迁移之后:

```
color = ColorKind.red
outline = OutlineKind.box
```

此更改影响下列属性:

- align
- color
- deviceType
- displayUse
- highlight
- indexOrientation
- intensity
- outline
- protect
- selectType
- sign

- 迁移工具将 **tableNames** 属性的值更改为字符串数组的数组。每个字符串数组必须有一到两个元素。第一个元素是表名，第二个元素（如果有的话）是表的标注。下表给出迁移工具如何更改 **tableNames** 属性的一些示例。

表 5. 对 **tableNames** 属性的更改

迁移之前	迁移之后
<code>{ tableNames = (table1, table2) }</code>	<code>{ tableNames = ["table1"], ["table2"] }</code>
<code>{ tableNames = (table1 t1, table2) }</code>	<code>{ tableNames = ["table1", "t1"], ["table2"] }</code>
<code>{ tableNames = (table1 t1, table2 t2) }</code>	<code>{ tableNames = ["table1", "t1"], ["table2", "t2"] }</code>

- 迁移工具会以更改 **tableNames** 属性值的方式来更改 **tableNameVariables** 属性值。
- 迁移工具将 **defaultSelectCondition** 属性的值更改为类型 `sqlCondition`。

迁移之前:

```
{ defaultSelectCondition =
  #sql{
    hostVar02 = 4
  }
}
```

迁移之后:

```
{ defaultSelectCondition =
  #sqlCondition{ // no space between #sqlCondition and the brace
    hostVar02 = 4
  }
}
```

- 迁移工具将 **fillCharacter** 的 NULL 值更改为空字符串值 ""。

相关任务

第 93 页的『将 EGL 代码迁移至 EGL 6.0 iFix』

相关概念

第 94 页的『EGL 至 EGL 迁移』

『设置 EGL 至 EGL 迁移首选项』

第 42 页的『日期、时间和时间戳记格式说明符』

设置 EGL 至 EGL 迁移首选项

可设置一些首选项，以控制 EGL V6.0 迁移工具转换 EGL 源代码的方式。有关迁移工具的更多信息，请参阅 *EGL 至 EGL 迁移*。为迁移工具设置首选项，如下所示：

1. 单击窗口 > 首选项。
2. 展开 **EGL**。
3. 单击 **EGL V6.0 迁移首选项**。

注： 如果找不到 **EGL V6.0 迁移首选项**，启用 EGL V6.0 迁移功能。请参阅 *启用 EGL 功能*。

4. 通过单击单选按钮来选择如何解决与新保留字的命名冲突：
 - **添加前缀** 设置迁移工具以将前缀添加至源代码中现在是保留字的所有单词。在文本框中使用此单选按钮，输入希望迁移工具添加至已更改单词的前缀。

- **添加后缀**设置迁移工具以将后缀添加至源代码中现在是保留字的所有单词。在文本框中使用此单选按钮，输入希望迁移工具添加至已更改单词的后缀。
5. 要将限定符添加至具有有限可用值的列表的属性值，选择**将限定符添加至枚举属性值**复选框。如果选择了此框，迁移工具会将值的类型添加至值的名称。
 6. 要将注释添加至迁移工具更改的文件，选择**记录级别**下面的某个选项。
 - **将注释添加至迁移工具处理的所有文件**会设置迁移工具以将注释添加至它处理的每个文件，即使它未更改该文件也是如此。
 - **将注释添加至迁移工具更改的所有文件**会设置迁移工具以将注释仅添加至它更改的文件。
 - **不要对文件添加注释**会设置迁移工具以便不对它处理的文件添加任何注释。
 - **添加至文件开头**会设置迁移工具以将注释添加至文件开头。
 - **添加至文件结尾**会设置迁移工具以将注释添加至文件结尾。
 7. 要将已处理文件的列表写至名为 `V60MigrationLog.txt` 的文件，选择**将迁移结果追加至每个项目的日志文件**复选框。
 8. 单击**应用**。
 9. 单击**确定**。

相关任务

第 113 页的『启用 EGL 功能』

第 93 页的『将 EGL 代码迁移至 EGL 6.0 iFix』

相关概念

第 94 页的『EGL 至 EGL 迁移』

设置环境

设置 EGL 首选项

设置基本 EGL 首选项，如下所示：

1. 单击窗口 > 首选项。
2. 列表显示后，单击 **EGL** 以显示 EGL 屏幕。
3. 选择或清除 **VisualAge Generator 兼容性** 的复选框。您的选择将影响在开发时可用的选项，如与 *VisualAge Generator* 的兼容性中所述。
4. 在**编码**列表框中，选择当创建新的 EGL 构建（.eglbld）文件时将使用的字符编码集。该设置对现有构建文件没有任何影响。缺省值为 UTF-8。
5. 在**用户标识**文本框中，指定用于访问远程构建机器（如果有的话）的用户标识。构建描述符选项 **destUserID** 具有优先权，并且该选项以及首选项值都优先于主构建描述符选项 **destUserID**。
6. 在**密码**文本框中，指定用于访问远程构建机器（如果有的话）的密码。构建描述符选项 **destPassword** 具有优先权，并且该选项以及首选项值都优先于主构建描述符选项 **destPassword**。
7. 单击应用。

要设置其它 EGL 首选项，请参阅本页底部的相关任务列表。设置首选项完成后，单击**确定**。

相关概念

第 295 页的『构建』

第 400 页的『与 VisualAge Generator 的兼容性』

相关任务

第 108 页的『设置缺省构建描述符』

第 106 页的『为 EGL 调试器设置首选项』

第 108 页的『设置源代码样式首选项』

第 110 页的『设置 SQL 数据库连接首选项』

第 112 页的『设置 SQL 检索首选项』

设置文本首选项

要更改 EGL 编辑器中显示文本的方式，执行下列操作：

1. 单击窗口 > 首选项。
2. 首选项列表显示后，展开**工作台**，然后单击**颜色和字体**。将显示“颜色和字体”窗格。
3. 展开 **EGL** 和**编辑器**，然后单击 **EGL 编辑器文本字体**。
4. 要从字体和颜色列表中进行选择，单击**更改按钮**，然后执行下列操作：
 - a. 要更改字体首选项，从可滚动列表中选择字体、字体样式和大小。
 - b. 要更改颜色首选项，从下拉列表中选择颜色。
 - c. 如果想要一条线从文本中间穿过，则选择**加删除线**复选框。

- d. 如果想要文本下方有一条线，则选择**加下划线**复选框。
- e. 可在“样本”框中见到所作选择的预览。选择完之后，单击**确定**。
5. 要使用缺省操作系统字体，单击**使用系统字体**按钮。
6. 要使用缺省工作台字体，单击**复位**按钮。
7. 要将所有编辑器（而不仅仅是 EGL 编辑器）的字体设置为缺省工作台字体，单击**恢复缺省值**按钮。
8. 要保存更改，单击**应用**或者（如果您完成了设置首选项）单击**确定**。

相关任务

第 105 页的『设置 EGL 首选项』

为 EGL 调试器设置首选项

要为 EGL 调试器设置首选项，遵循下列步骤：

1. 单击**窗口 > 首选项**。
2. 列表显示后，展开 **EGL** 并单击**调试**。
3. 清除或选择标记为**需要时提示 SQL 用户标识和密码**的复选框。

有关选项的详细信息，请参阅 *EGL 调试器*。

4. 清除或选择标记为将 **systemType** 设置为 **DEBUG** 的复选框。

有关选项的详细信息，请参阅 *EGL 调试器*。

5. 设置 `sysVar.terminalID`、`sysVar.sessionID` 和 `sysVar.userID` 的初始值。如果未指定值，则每个值都缺省为 Windows 2000/NT/XP 或 Linux™ 上的用户标识。
6. 设置“EGL 调试器端口”值。缺省端口是 8345。
7. 选择要在调试会话期间处理数据时使用的字符编码类型。缺省值是本地系统的文件编码。有关选项的详细信息，请参阅 *EGL 调试器的字符编码选项*。
8. 要指定外部 Java 类以便在调试器运行时使用，请修改类路径。可能需要额外的类，以便支持（例如）MQSeries、JDBC 驱动程序或 Java 访问函数。

类路径附加内容对 WebSphere® Application Server 测试环境是不可视的；但可以通过使用服务器配置的“环境”选项卡来对该环境的类路径进行添加。

使用“类路径顺序”框右边的按钮：

- 要添加项目、JAR 文件、目录或变量，单击适当的按钮：**添加项目**、**添加 JAR**、**添加目录**或**添加变量**。
 - 要除去一个条目，选择该条目，并单击**除去**。
 - 要在两个或多个条目组成的列表中移动一个条目，选择该条目，并单击**上移**或**下移**。
9. 要恢复缺省设置，单击**恢复缺省值**。
 10. 要保存更改，单击**应用**或者（如果您完成了设置首选项）单击**确定**。

相关概念

第 400 页的『与 VisualAge Generator 的兼容性』

第 255 页的『EGL 调试器』

第 107 页的『EGL 调试器的字符编码选项』

相关任务

第 110 页的『设置 SQL 数据库连接首选项』

相关参考

第 856 页的『sessionID』

第 860 页的『terminalID』

第 861 页的『userID』

EGL 调试器的字符编码选项

EGL 调试器允许您指定要在调试时使用的字符编码类型。字符编码控制调试器如何在内部表示字符和数字数据、如何比较字符数据以及如何将参数传递至远程程序、文件和数据库。要更改这些选项，请参阅设置 *EGL 调试器的首选项*。

EGL 调试器支持两种不同类型的字符编码：本地系统上的缺省编码和 EBCDIC。EGL 调试器的缺省字符编码与本地系统的缺省编码相同。

- 选择缺省字符编码后，调试器会用缺省格式（通常是 ASCII）表示 CHAR、DBCHAR、MBCHAR、DATE、TIME、INTERVAL、NUM 和 NUMC 变量。字符变量之间的比较使用 ASCII 整理顺序。当调用远程程序和访问远程文件和数据库时，必须将数据转换为主机格式。

如果选择此设置并且不指定转换表，调试器会在您调用远程程序或访问远程文件或数据库时选择相应的转换表。有关转换表的更多信息，请参阅 *数据转换*。

- 使用 EBCDIC 字符编码时，调试器会用 EBCDIC 编码表示 CHAR、DBCHAR、MBCHAR、DATE、TIME 和 INTERVAL 变量。NUM 和 NUMC 变量是用主机数字格式表示的。字符变量之间的比较使用 EBCDIC 整理顺序。在调用远程程序或访问远程文件和数据库时，不必将数据转换为主机格式，但在进行 SQL 调用或对本地 C++ 例程进行调用时，数据将转换为相应的 Java 或 ASCII 格式。EBCDIC 编码有若干语言版本可用。

如果选择 EBCDIC 字符编码并且不指定转换表，在您调用远程程序或访问远程文件或数据库时，调试器不会使用转换表。程序名、库名和所有传递参数是根据 EBCDIC 字符编码进行编码的。

如果 Java 运行时环境不支持所选字符编码，则在调试器启动时会显示一条警告消息。如果选择继续调试，调试器将返回至缺省编码类型。

在调试会话期间，不能更改字符编码。必须重新启动调试器以便让字符编码的更改生效。

相关概念

第 255 页的『EGL 调试器』

相关任务

第 106 页的『为 EGL 调试器设置首选项』

相关参考

第 426 页的『数据转换』

设置缺省构建描述符

有关缺省构建描述符和构建描述符优先顺序规则的概述，请参阅在工作台中生成。

要在工作台级别上指定构建描述符首选项，执行下列操作：

1. 单击**窗口 > 首选项**。
2. 列表显示后，展开 **EGL** 并单击**缺省构建描述符**。
3. 选择“调试”构建描述符和“目标”系统构建描述符。
4. 单击**应用**，然后单击**确定**。

要在文件、文件夹、包或项目级别上指定构建描述符首选项，执行下列操作：

1. 右键单击您感兴趣的级别（例如，右键单击文件或文件夹名称）并从上下文菜单中单击**属性**。
2. 选择 **EGL 缺省构建描述符**。
3. 选择“调试”构建描述符和“目标”系统构建描述符。
4. 单击**确定**。

相关概念

第 300 页的『工作台中的生成』

设置 EGL 编辑器的首选项

要指定 EGL 编辑器首选项，执行下列操作：

1. 单击**窗口 > 首选项**。
2. 列表显示后，展开 **EGL** 并单击**编辑器**。
3. 要在查看 EGL 文件时显示行号，选择**显示行号**复选框。要清除行号，清除该复选框。文件本身不受影响。
4. 要在源代码中找到错误的位置显示红色下划线，选择**注释文本中的错误**复选框。要清除那些下划线，清除该复选框。文件本身不受影响。
5. 要在源代码中找到错误的位置在编辑器右页边距（概述标尺）中显示红色错误指示符，选择**在概述标尺中注释错误**复选框。单击错误指示符将使您转到源代码中的出错位置。要清除错误指示符，清除该复选框。文件本身不受影响。
6. 要指定源代码样式，请完成**设置源代码样式首选项**中描述的过程。
7. 要添加、除去和定制用于内容辅助的模板，请完成**设置模板首选项**中描述的过程。
8. 要更改文本显示方式，遵循**设置文本首选项**中描述的过程。

相关任务

『设置源代码样式首选项』

第 109 页的『设置模板首选项』

第 105 页的『设置文本首选项』

相关参考

第 441 页的『EGL 中的内容辅助』

设置源代码样式首选项

可以更改 EGL 代码在 EGL 编辑器中的显示方式：

1. 单击**窗口 > 首选项**

2. 首选项列表显示后，展开 **EGL** 和 **编辑器**，然后单击**源代码样式**。
3. 要选择要在源类型背后显示的颜色，请单击“背景色”框中的**定制**单选按钮。单击“定制”标签旁边的按钮。将显示颜色调色板。选择一种颜色，然后单击**确定**。
4. 在“前景”框中，选择文本类型，然后单击**颜色**按钮。将显示颜色调色板。选择一种颜色，然后单击**确定**。
5. 如果想要类型为**粗体**，则选择**粗体**复选框。
6. 要保存更改，单击**应用**或者（如果您完成了设置首选项）单击**确定**。

相关任务

第 105 页的『设置 EGL 首选项』

设置模板首选项

执行下列操作以添加、除去或定制当您在 EGL 编辑器中请求内容辅助时显示的模板：

1. 单击**窗口 > 首选项**。
2. 首选项列表显示后，展开 **EGL** 和 **编辑器**，然后单击**模板**。将显示模板列表。

注：就象在 Windows 2000/NT/XP 上的其它应用程序中一样，可以通过单击条目来选择它；可以使用 **Ctrl-单击**来选择或取消选择条目而不影响其它选择；并且可以使用 **Shift-单击**来选择一组条目（一直到您最后单击的条目）。

3. 要使模板在 EGL 编辑器中可用，请选择模板名称左边的复选框。要使列示的所有模板都可用，请单击**全部启用**。同样，要使模板不可用，请清除相关的复选框；要使列示的所有模板都不可用，请单击**全部禁用**。
4. 要创建新的模板，执行下列操作：

a. 单击**新建**

- b. “新建模板”对话框显示后，由于要保证仅当名称与描述的组合在所有模板中都唯一时才在内容辅助列表中显示模板，所以请同时指定名称和描述。

注：如果模板中使用的第一个字是 EGL 关键字（如 **Function**），则当您在 EGL 编辑器中请求内容辅助时，该模板可用，但仅当屏幕上的光标位于该字有效的位置时才会这样。同样，如果输入前缀紧接着请求内容辅助，则将提供所有以该前缀开头的模板（倘若屏幕上的光标位于在语法上允许该模板的位置）。例如，输入“fun”以请求函数模板。如果未输入前缀或完整的第一个字，则当您请求内容辅助时您将看不到任何模板。

c. 在**模式**字段中，输入模板本身：

- 输入要显示的任何文本
- 要在屏幕光标位置放置一个预先存在的变量，请单击**插入变量**，然后双击一个变量。当在 EGL 编辑器中插入模板时，那些变量中的每一个变量都解析为适当的值。
- 要创建定制变量，请输入美元符号（\$），后跟左花括号、字符串和右花括号，如以下示例所示：

```
 ${variable}
```

您可能会发现，比较简单的方法是插入预先存在的变量并更改名称以供您自己使用。

当在 EGL 编辑器中插入定制模板时，每个变量都带有下划线，以指示值是必需的。

- 要完成任务，请单击**确定**，并在模板屏幕上单击**应用**。
- 5. 要查看现有的模板，请单击列示的条目并查看“预览”框。
- 6. 要编辑现有的模板，请单击列示的条目，然后单击**编辑**。与“编辑模板”对话框进行交互，就象对“新建模板”对话框所做的那样。
- 7. 要除去现有的模板，请单击列示的条目，然后单击**除去**。要除去多个模板，请使用用于选择多个列表条目的 Windows 2000/NT/XP 约定，然后单击**除去**。
- 8. 要从 XML 文件中导入模板，请单击模板列表右边的**导入**并遵循浏览机制以指定文件位置。
- 9. 要将模板导出至 XML 文件，请单击模板列表右边的**导出**并遵循浏览机制以指定新文件的位置。要导出多个模板，请使用用于选择多个列表条目的 Windows 2000/NT/XP 机制，然后单击**导出**。
- 10. 要将列示的所有模板都导出至 XML 文件，请单击**全部导出**并遵循浏览机制以指定文件位置。
- 11. 要保存更改，请单击**应用**。要返回至安装时有效的模板列表，请单击**恢复缺省值**。

相关任务

第 118 页的『将 EGL 模板与内容辅助配合使用』

设置 SQL 数据库连接首选项

您由于下列原因而使用 SQL 数据库连接页：

- 可以对在 J2EE 外部访问的数据库启用声明时和调试时访问。
- 并且，可以对构建描述符选项 `sqlJNDIName` 设置一个值，该选项指定在 JNDI 注册表中与缺省数据源绑定的名称；例如，`java:comp/env/jdbc/MyDB`。该选项位于在下列情况下为您创建的构建描述符中：
 - 您使用“EGL Web 项目”向导，如**创建项目以使用 EGL**所述；并且
 - 当在该向导中工作时，您请求创建构建描述符。

执行下列操作：

1. 单击**窗口 > 首选项**
2. 首选项列表显示后，展开 **EGL**，然后单击 **SQL 数据库连接**。
3. 在**连接 URL** 字段中，输入用于通过 JDBC 连接至数据库的 URL：
 - 对于 IBM DB2 APP DRIVER for Windows，URL 是 `jdbc:db2:dbName`（其中 `dbName` 是数据库名称）
 - 对于 Oracle JDBC 瘦客户端驱动程序，URL 根据数据库位置而变化。如果数据库相对于机器来说是本地的，则 URL 为 `jdbc:oracle:thin:dbName`（其中 `dbName` 是数据库名称）。如果数据库在远程服务器上，则 URL 为 `jdbc:oracle:thin:@host:port:dbName`（其中 `host` 是数据库服务器的主机名，`port` 是端口号，`dbName` 是数据库名称）
 - 对于 Informix JDBC NET 驱动程序，URL 如下所示（应将下列各行组成一行）：

```
jdbc:informix-sqli://host:port  
/dbName:informixserver=servername;  
user=userName;password=passWord
```

host

数据库服务器所在的机器的名称

port

端口号

dbName

数据库名称

serverName

数据库服务器的名称

userName

Informix 用户标识

passWord

与用户标识相关联的密码

4. 在**数据库**字段中，输入数据库的名称。
5. 在**用户标识**字段中，输入用来进行连接的用户标识。
6. 在**密码**字段中，输入该用户标识的密码。
7. 在**数据库供应商类型**字段中，选择正在用于 JDBC 连接的数据库产品和版本。
8. 在 **JDBC 驱动程序**字段中，选择正在用于 JDBC 连接的 JDBC 驱动程序。
9. 在 **JDBC 驱动程序类**字段中，输入所选择的驱动程序的驱动程序类。对于 IBM DB2 APP DRIVER for Windows，驱动程序类为 COM.ibm.db2.jdbc.app.DB2Driver；对于 Oracle JDBC 瘦客户机端驱动程序，驱动程序类为 oracle.jdbc.driver.OracleDriver；对于 Informix JDBC NET 驱动程序，驱动程序类为 com.informix.jdbc.IfxDriver。对于其它驱动程序类，参阅驱动程序的文档。
10. 在**类位置**字段中，输入包含驱动程序类的 *.jar 或 *.zip 文件的标准文件名。对于 IBM DB2 APP DRIVER for Windows，输入 db2java.zip 文件的标准文件名；例如，d:\sql11b\java\db2java.zip。对于 Oracle THIN JDBC DRIVER，输入 classes12.zip 文件的标准文件名；例如，d:\ora81\jdbc\lib\classes12.zip。对于其它驱动程序类，参阅驱动程序的文档。
11. 在**连接 JNDI 名称**字段中，指定在 J2EE 中使用的数据库。该值是在 JNDI 注册表中与数据源绑定的名称；例如，java:comp/env/jdbc/MyDB。如之前所述，在为给定 EGL Web 项目自动构造的构建描述符中，此值被赋值给选项 **sqlJNDIName**。
12. 如果您正在访问 DB2 UDB 并在**辅助认证标识**字段中指定一个值，则 EGL 在验证时使用的 SET CURRENT SQLID 语句将使用该值。该值区分大小写。

可以清除或应用首选项设置：

- 要恢复缺省值，单击**恢复缺省值**。
- 要应用首选项设置而不退出首选项对话框，单击**应用**。
- 如果完成了设置首选项，则单击**确定**。

相关任务

第 116 页的『创建 EGL Web 项目』

第 105 页的『设置 EGL 首选项』

相关参考

第 365 页的『sqlJNDIName』

设置 SQL 检索首选项

在 EGL 声明时，可以使用 SQL 检索功能来根据 SQL 表的各列来创建 SQL 记录。有关概述，请参阅 *SQL 支持*。

要为 SQL 检索功能设置首选项，执行下列操作：

1. 单击窗口 > 首选项，然后展开 **EGL**，并单击 **SQL 检索**
2. 指定用于创建 SQL 检索功能创建的每个结构项的规则：
 - a. 要指定在从 SQL 字符数据类型创建结构项时要使用的 EGL 类型，单击下列其中一个单选按钮：
 - 使用 **EGL 类型字符串**（缺省值）将 SQL 字符型数据类型映射至 EGL 字符串数据类型
 - 使用 **EGL 类型字符型**将 SQL 字符型数据类型映射至 EGL 字符型数据类型
 - 使用 **EGL 类型 mbChar** 将 SQL 字符型数据类型映射至 EGL mbChar 数据类型
 - 使用 **EGL 类型 Unicode** 将 SQL 字符型数据类型映射至 EGL Unicode 数据类型
 - b. 要指定结构项名的大小写，单击下列单选按钮之一：
 - **不更改大小写**（缺省值）意味着结构项名的大小写与相关表列名的大小写相同
 - **更改为小写**意味着结构项名是表列名的小写版本
 - **更改为小写并大写下划线后面的第一个字母**也意味着除了结构项名中有一个字母显示为大写之外（如果在表列名中该字母紧跟在下划线后面），结构项名是表列名的小写版本
 - c. 要指定在结构项名中如何反映表列名中的下划线，单击下列单选按钮之一：
 - **不更改下划线**（缺省值）意味着将表列名中的下划线包括在结构项名中
 - **除去下划线**意味着不将表列名中的下划线包括在结构项名中
 - **将下划线更改为连字符**意味着表列名中的下划线在结构项名中显示为连字符
3. 如果您打算检索的是 Informix 系统模式的一部分的表中的数据，则清除**排除系统模式**复选框。（在这种情况下，“Informix”是表所有者。）在所有其它情况下，选择该复选框来提高 SQL 检索功能的性能。

缺省情况下，已选中该复选框。

相关概念

第 209 页的『SQL 支持』

相关任务

第 231 页的『检索 SQL 表数据』

第 105 页的『设置 EGL 首选项』

第 110 页的『设置 SQL 数据库连接首选项』

相关参考

第 230 页的『Informix 和 EGL』

启用 EGL 功能

为了访问 EGL 功能，必须启用 EGL 功能。下列 EGL 功能可用：

EGL 开发

包括与开发和调试 EGL 应用程序有关的所有功能。

EGL V6.0 迁移

包括与转换 EGL 5.1.2 和 6.0 源代码以符合 EGL V6.0 iFix 有关的所有功能。

VisualAge Generator 至 EGL 迁移

包括将现有 VisualAge Generator 代码迁移至 EGL 代码的所有功能。

要启用 EGL 功能，执行下列操作：

1. 单击**窗口 > 首选项**。
2. 首选项列表显示后，展开**工作台**，然后单击**功能**。将显示“功能”窗格。
3. 如果某个功能部件需要已启用功能并且您希望在第一次使用该功能部件时接收提示，则选择在**启用功能时提示**复选框。
4. 展开 **EGL Developer** 功能文件夹。
5. 选择期望的 EGL 功能的复选框。或者可以选择 **EGL Developer** 功能文件夹来启用该文件夹包含的所有功能。
6. 要将已启用功能列表设置回安装产品时的状态，单击**复原缺省值**按钮。
7. 要保存更改，单击**应用**，然后单击**确定**。

注：启用 EGL 功能将自动启用开发和调试 EGL 应用程序所需的任何其它功能。

相关任务

第 105 页的『设置 EGL 首选项』

开始开发代码

创建项目

创建 EGL 项目

有关如何组织工作的概述，请参阅 *EGL 项目、包和文件*。

要设置新的 EGL 项目，执行下列操作：

1. 在工作台中，执行下列任何一个步骤：

- 单击 **文件 > 新建 > 项目**；或者
- 单击鼠标右键，然后单击 **新建 > 项目**。

“新建项目”向导将打开。

2. 展开 **EGL**，然后单击 **EGL 项目**。单击下一步。**新建 EGL 项目** 向导显示。

注：如果 EGL 项目不可用，则选择 **显示所有向导** 复选框。

3. 在 **项目名称** 字段中，输入项目的名称。缺省情况下，项目将被放在工作空间中，但您可以单击 **浏览** 并选择另一个位置。

4. 选择如何指定构建描述符，该构建描述符是在生成时控制处理的部件：

- **自动创建新的项目构建描述符** 表示 EGL 提供构建描述符并将它们写至与项目同名的构建文件（扩展名为 `.eglbld`）。

要在那些构建描述符中指定一些值，请单击 **选项**。以后，要更改那些值，请更改为您创建的构建文件。

有关更多详细信息，请参阅 *在创建项目时指定数据库选项*。

- **使用 EGL 首选项中指定的构建描述符** 表示 EGL 指向作为 EGL 首选项创建并标识的构建描述符。
 - **选择现有的构建描述符** 允许您从工作空间中的那些可用构建描述符中指定构建描述符。
5. 在大多数情况下，单击 **完成**。但是，如果单击 **下一步**，则可以指定要从正在创建的项目中引用的其它源文件夹和项目。当您完成对其它源文件夹和项目的选择之后，单击 **完成**。

相关概念

第 267 页的『构建描述符部件』

第 13 页的『EGL 项目、包和文件』

相关任务

第 117 页的『在创建项目时指定数据库选项』

第 110 页的『设置 SQL 数据库连接首选项』

创建 EGL Web 项目

有关如何组织工作的概述，请参阅 *EGL 项目、包和文件*。

要设置新的 EGL Web 项目，执行下列操作：

1. 在工作台中，执行下列任何一个步骤：

- 单击 **文件 > 新建 > 项目**；或者
- 单击鼠标右键，然后单击 **新建 > 项目**。

“新建项目”向导将打开。

2. 展开 **EGL**，然后单击 **EGL Web 项目**。单击下一步。**新建 EGL Web 项目** 向导将显示。

3. 在 **项目名称** 字段中，输入项目的名称。缺省情况下，项目将被放在工作空间中；但您可以单击 **浏览** 并选择另一个位置。

4. 选择如何指定构建描述符，该构建描述符是在生成时控制处理的部件：

- **自动创建新的项目构建描述符** 表示 EGL 提供构建描述符并将它们写至与项目同名的构建文件（扩展名为 `.eglbld`）。

要在那些构建描述符中指定一些值，请单击 **选项**。以后，要更改那些值，请更改为您创建的构建文件。

有关更多详细信息，请参阅 *在创建项目时指定数据库选项*。

- **使用 EGL 首选项中指定的构建描述符** 表示 EGL 指向作为 EGL 首选项创建并标识的构建描述符。
- **选择现有的构建描述符** 允许您从工作空间中的那些可用构建描述符中指定构建描述符。

5. 如果已请求自动创建构建描述符，则可以在用于 **SQL 连接的 JNDI 名称** 字段中填写一个值。其作用是指定在调试或生成时 JNDI 注册表中缺省数据源所绑定的名称。（一个示例值是 `java:comp/env/jdbc/MyDB`。）您的选择将对构建描述符选项 `sqlJNDIName` 赋值。如果用于 **SQL 连接的 JNDI 名称** 字段已被填充，则该值是从工作台首选项中获取的，如设置 *SQL 数据库连接首选项* 中所述。

6. 在大多数情况下，单击 **完成**。要执行附加定制（这对于任何 Web 项目都是有可能的），在对话框的底部配置 J2EE 设置。可选择单击 **隐藏高级设置** 以取消 J2EE 设置。单击 **下一步**。选择功能设置，然后单击 **下一步**。选择页面模板，然后单击 **完成**。

相关概念

第 267 页的『构建描述符部件』

第 13 页的『EGL 项目、包和文件』

相关任务

第 117 页的『在创建项目时指定数据库选项』

第 110 页的『设置 SQL 数据库连接首选项』

相关参考

第 365 页的『`sqlJNDIName`』

在创建项目时指定数据库选项

要在 EGL 自动创建的构建描述符中指定选项值，请使用项目构建选项对话框。有关如何显示该对话框的详细信息，请参阅[创建项目以使用 EGL](#)。

要接受首选项中指定的数据库连接信息，请单击复选框。

下表显示了屏幕上的每个标签以及相关构建描述符选项。

标号	构建描述符选项
数据库类型	dbms
数据库 JDBC 驱动程序	sqlJDBCDriverClass
数据库名称	sqlJNDIName (对于 J2EE 输出) 或 sqlDB (对于非 J2EE 输出)

相关概念

第 267 页的『构建描述符部件』

相关任务

第 116 页的『创建 EGL Web 项目』

相关参考

第 347 页的『构建描述符选项』

创建 EGL 源文件夹

一旦在工作台中创建了项目，就可以在该项目中创建一个或多个文件夹以包含 EGL 文件。

要创建用于对 EGL 文件进行分组的文件夹，执行下列操作：

1. 在工作台中，单击**文件 > 新建 > EGL 源文件夹**。
2. 选择将包含 EGL 文件夹的项目。在“文件夹名”字段中，输入 EGL 文件夹的名称，例如，myFolder。
3. 单击**完成**按钮。

相关概念

第 13 页的『EGL 项目、包和文件』

第 1 页的『EGL 简介』

相关任务

第 116 页的『创建 EGL Web 项目』

相关参考

第 118 页的『创建 EGL 源文件』

第 612 页的『命名约定』

创建 EGL 包

EGL 包是相关源部件的已命名集合。要创建 EGL 包，执行下列操作：

1. 标识用来包含包的项目或文件夹。如果还没有项目或文件夹，则必须创建项目或文件夹。
2. 在工作台中，单击**文件 > 新建 > EGL 包**。
3. 选择将包含 EGL 包的项目或文件夹。“源文件夹”字段可能已根据“项目资源管理器”中的当前选择进行预先填充。
4. 在“包名”字段中，输入 EGL 包的名称。有关包命名约定的详细信息，请参阅 *EGL 项目、包和文件*。
5. 单击**完成**按钮。

相关概念

第 13 页的『EGL 项目、包和文件』

第 1 页的『EGL 简介』

相关任务

第 117 页的『创建 EGL 源文件夹』

第 116 页的『创建 EGL Web 项目』

相关参考

『创建 EGL 源文件』

创建 EGL 源文件

要创建 EGL 源文件，执行下列操作：

1. 标识用来包含文件的项目或文件夹。如果还没有项目或文件夹，则必须创建项目或文件夹。
2. 在工作台中，单击**文件 > 新建 > EGL 源文件**。
3. 选择将包含 EGL 文件的项目或文件夹。选择将包含 EGL 文件的包。在“EGL 源文件名”字段中，输入 EGL 文件的名称，例如，myEGLFile。
4. 单击**完成**以创建文件。将把扩展名 (.egl) 自动追加到文件名的末尾。EGL 文件便出现在“项目资源管理器”视图中，并在缺省 EGL 编辑器中自动打开。

相关概念

第 13 页的『EGL 项目、包和文件』

第 1 页的『EGL 简介』

相关任务

第 117 页的『创建 EGL 源文件夹』

第 116 页的『创建 EGL Web 项目』

将 EGL 模板与内容辅助配合使用

要使用内容辅助进行练习，执行下列操作：

1. 打开新的 EGL 文件。
2. 在可用的行上，输入 **P**（表示 PageHandler 或程序）并按 **Alt + /**。
3. 当弹出菜单显示时，单击要进行定制的部件的图标。执行下列其中一个步骤：
 - 按 **Enter** 键以选择列表中的第一个图标；或者
 - 使用方向键来选择另一个图标（对于程序）并按 **Enter** 键。

编辑器将把部件模板放入文件。

4. 定制该部件。

当模板显示时，编辑器将突出显示第一个需要输入信息的区域；在此例中，请指定部件名。输入之后，按 **Tab** 键以突出显示下一个需要输入信息的区域。

可以重复使用 **Tab** 键，在到达文件结尾或通过任何其它方式更改了文件中的位置之前，此键一直可用。

5. 要将函数插入到程序或 PageHandler 中，请输入 **F**（表示函数），然后按 **Alt + /**。尽管您可以再次选择部件模板，但请执行下列操作：

- 使用方向键或鼠标来滚动至列表结尾
- 按 **Enter** 键或单击字 *Function*；注意，没有图标意味着您正在选择字符串而不是部件模板

选择字符串的能力在其它上下文中（例如，当您想要快速输入变量名时）更为有用。

6. 将光标定位在字 *Function* 的结尾，按 **Alt + /** 并单击列表中的图标。

编辑器将把该函数模板放入文件。

7. 定制该部件。

8. 开发代码时，请定时按 **Alt + /** 以了解所提供的服务的范围。

相关任务

第 137 页的『将代码段插入到 EGL 和 JSP 文件中』

第 109 页的『设置模板首选项』

相关参考

第 481 页的『EGL 源格式的函数部件』

第 619 页的『EGL 源格式的 PageHandler 部件』

第 664 页的『EGL 源格式的程序部件』

EGL 的键盘快捷键

下表显示了在 EGL 编辑器中可用的键盘快捷键。

键组合	功能
Ctrl+/	注释
Ctrl+\	取消注释
Ctrl+A	全部选中
Ctrl+C	复制
Ctrl+F	查找
Ctrl+H	搜索
Ctrl+K	查找下一个
Ctrl+S	保存
Ctrl+V	粘贴
Ctrl+X	剪切

键组合	功能
Ctrl+G	生成
Ctrl+L	转至特定行
Ctrl+Y	重做
Ctrl+Z	撤销
Ctrl+Shift+A	将显式 SQL 语句添加至具有隐式 SQL 语句的 EGL I/O 语句
Ctrl+Shift+K	查找上一个
Ctrl+Shift+N	访问“打开部件”对话框
Ctrl+Shift+P	构造 EGL prepare 语句和相关的 get 、 execute 或 open 语句
Ctrl+Shift+R	使用检索功能来创建或覆盖 SQL 记录部件中的项
Ctrl+Shift+S	在“项目资源管理器”中显示当前文件
Ctrl+Shift+V	查看和验证与 EGL I/O 语句相关联的 SQL 语句并执行相关操作
Ctrl+空格键	获取内容辅助
F3	打开包含突出显示名称的部件的文件
Tab 键	使文本缩进到下一个制表符停止位

开发基本 EGL 源代码

创建 EGL dataItem 部件

EGL dataItem 部件定义不能细分的内存区。EGL dataItem 部件包含在 EGL 文件中。要创建 EGL dataItem 部件，执行下列操作：

1. 标识要包含 dataItem 部件的 EGL 文件并在 EGL 编辑器中打开该文件。如果还没有 EGL 文件，则必须创建它。
2. 根据 EGL 语法输入 dataItem 部件的细节（有关详细信息，请参阅 *EGL 源代码格式的 DataItem 部件*）。可使用内容辅助以将 dataItem 部件语法的概要内容放在文件中。
3. 保存 EGL 文件。

相关概念

『DataItem 部件』

第 13 页的『EGL 项目、包和文件』

相关任务

第 118 页的『创建 EGL 源文件』

第 118 页的『将 EGL 模板与内容辅助配合使用』

相关参考

第 441 页的『EGL 中的内容辅助』

第 431 页的『EGL 源格式的 DataItem 部件』

第 612 页的『命名约定』

DataItem 部件

dataItem 部件定义不能被细分的内存区。*dataItem* 部件是一个独立的部件，这与固定结构中的结构字段不同。

基本变量是基于 *dataItem* 部件或基本声明（如 INT 或 CHAR(2)）的内存区。可按照下列方式使用基本变量：

- 作为将数据接收到函数或程序中的参数
- 作为 EGL 函数中的变量；例如，在赋值语句中使用，或作为将数据传递给另一函数或程序的自变量

每个基本变量都具有一系列属性，这些属性或者是缺省属性，或者是在变量或 *dataItem* 部件中指定的。有关详细信息，请参阅 *EGL 属性与覆盖概述*。

相关概念

第 123 页的『固定记录部件』

第 23 页的『固定结构』

第 59 页的『EGL 属性概述』

第 16 页的『部件』

第 122 页的『记录部件』

第 25 页的『Typedef』

相关任务

第 109 页的『设置模板首选项』

相关参考

第 431 页的『EGL 源格式的 DataItem 部件』

第 448 页的『EGL 源格式』

第 430 页的『数据初始化』

第 31 页的『基本类型』

创建 EGL 记录部件

记录部件定义了结构（存储器中固定大小的数据元素的分层布局）以及记录和外部数据源关系的可选绑定，这些数据源包括文件、数据库或消息队列。EGL 记录部件包含在 EGL 文件中。要创建 EGL 记录部件，执行下列操作：

1. 标识要包含记录部件的 EGL 文件并在 EGL 编辑器中打开该文件。如果还没有 EGL 文件，则必须创建它。
2. 根据 EGL 语法输入记录部件的细节（有关详细信息，请参阅 *EGL 源代码格式的基本记录部件*、*EGL 源代码格式的带索引记录部件*、*EGL 源代码格式的 MQ 记录部件*、*EGL 源代码格式的相关记录部件*、*EGL 源代码格式的串行记录部件*和 *EGL 源代码格式的 SQL 记录部件*）。可使用内容辅助以将记录部件语法的概要内容放在文件中。
3. 保存 EGL 文件。

相关概念

第 13 页的『EGL 项目、包和文件』

『记录部件』

相关任务

第 118 页的『创建 EGL 源文件』

第 118 页的『将 EGL 模板与内容辅助配合使用』

相关参考

第 345 页的『EGL 源格式的基本记录部件』

第 441 页的『EGL 中的内容辅助』

第 488 页的『EGL 源格式的带索引记录部件』

第 603 页的『EGL 源格式的 MQ 记录部件』

第 612 页的『命名约定』

第 676 页的『EGL 源格式的相关记录部件』

第 679 页的『EGL 源格式的串行记录部件』

第 683 页的『EGL 源格式的 SQL 记录部件』

记录部件

记录部件定义数据序列，在生成时不一定要知道它的长度，它的内容由字段组成。在 EGL 中，字段定义基于记录部件的任何记录中的变量。

字段可以是字典、arrayDictionary 或一组字典或一组 arrayDictionary；或者可以基于下列任何一项：

- 基本类型，如 STRING

- DataItem 部件
- 固定记录部件（如下所述）
- 另一个记录部件
- 任何先前种类的数组

有两种类型的记录部件:

- basicRecord, 用于一般处理, 而不是用于访问数据存储
- SQLRecord, 用于访问关系数据库

可在下列上下文中使用记录:

- 在将数据复制至关系数据库或从关系数据库复制数据的语句中
- 在赋值或移动语句中
- 作为将数据传递至另一程序或函数的自变量
- 作为将数据接收到程序或函数中的参数

固定记录部件与记录部件截然不同, 固定记录部件定义长度在生成时已知的数据序列。固定记录部件主要用于访问 VSAM 文件、MQSeries 消息队列和其它顺序文件。

包括级别号的记录部件就是固定记录部件, 即使该记录部件的类型为 basicRecord 或 SQLRecord 也是如此。有关其它详细信息, 请参阅*固定记录部件*。

相关概念

- 第 121 页的『DataItem 部件』
- 『固定记录部件』
- 第 16 页的『部件』
- 第 125 页的『记录类型和属性』
- 第 277 页的『资源关联和文件类型』
- 第 23 页的『固定结构』
- 第 25 页的『Typedef』

相关任务

- 第 108 页的『设置缺省构建描述符』
- 第 108 页的『设置 EGL 编辑器的首选项』

相关参考

- 第 448 页的『EGL 源格式』
- 第 430 页的『数据初始化』
- 第 31 页的『基本类型』

固定记录部件

固定记录部件定义长度在生成时已知的数据序列。此类型的部件必定由一系列基本的固定长度字段组成, 每个字段可以是具有子结构的。例如, 可按如下所示定义指定电话号码的字段:

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

尽管您可以将固定记录（它们是变量）用于各种类型的处理，但它们最好的用途还是用于 VSAM 文件、MQSeries 消息队列和其它顺序文件的 I/O 操作上。尽管可将固定记录用于访问关系数据库或一般处理（如用于 VisualAge Generator 之类的先前产品），但还是建议您避免在新的开发中将固定记录用于这些用途。

下列任何类型的记录部件都是固定记录类型：

- indexedRecord
- mqRecord
- relationalRecord
- serialRecord

此外，如果每个字段前存在级别号，则下列任何类型的记录部件都是固定记录部件：

- basicRecord
- SQLRecord

可在下列上下文中使用固定记录：

- 在将数据复制至数据源或从数据源复制数据的语句中
- 在赋值或移动语句中
- 作为将数据传递至另一程序或函数的自变量
- 作为将数据接收到程序或函数中的参数

固定记录部件与外部数据源之间的关系由固定记录部件的类型和一组特定于类型的属性（如 fileName）确定。例如，基于类型 indexedRecord 的记录用于访问 VSAM 键序列数据集。记录部件与数据源之间的关系确定在 EGL I/O 语句（如 **add**）中使用固定记录时生成的操作。

固定记录字段可基于另一固定记录部件；在赋值语句中，该字段被视作类型为 CHAR 的内存区，不管固定记录部件中的类型如何都是如此。

相关概念

- 第 121 页的『DataItem 部件』
- 第 122 页的『记录部件』
- 第 125 页的『记录类型和属性』
- 第 277 页的『资源关联和文件类型』
- 第 23 页的『固定结构』
- 第 25 页的『Typedef』

相关任务

- 第 108 页的『设置缺省构建描述符』
- 第 108 页的『设置 EGL 编辑器的首选项』

相关参考

- 第 340 页的『赋值』
- 第 448 页的『EGL 源格式』
- 第 430 页的『数据初始化』
- 第 31 页的『基本类型』

记录类型和属性

您可以使用多种 EGL 记录类型:

- `basicRecord`
- `indexedRecord`
- `mqRecord`
- `relativeRecord`
- `serialRecord`
- `SQLRecord`

有关哪些目标系统支持哪些记录类型的详细信息, 请参阅[记录和文件类型交叉引用](#)。
有关如何初始化记录部件的详细信息, 请参阅[数据初始化](#)。

basicRecord

基本记录或固定基本记录用于内部处理, 它们不能访问数据存储器。

在缺省情况下, 该部件是一个记录部件, 但如果字段定义前有级别号, 则该部件是一个固定记录部件。

在类型为 `basicRecord` 的固定记录部件中, 属性 **redefines** 是可用的。如果设置了该属性, 则该属性标识已声明记录, 而基于该固定记录部件的任何记录将访问已声明记录的运行时内存。

如[数据初始化](#)中所述, 在主程序中, 程序属性 **inputRecord** 标识自动初始化的记录 (或固定记录)。

indexedRecord

带索引记录是一个固定记录, 它允许您使用通过键值 (它标识文件中记录的逻辑位置) 访问的文件。可以通过调用 **get**、**get next** 或 **get previous** 语句来读取该文件。并且, 也可以通过调用 **add** 或 **replace** 语句来写入文件; 并且可以通过调用 **delete** 语句来从该文件中除去记录。

类型为 `indexedRecord` 的部件的属性包括下列各项:

- **fileName** 是必需的。有关输入的含义的详细信息, 请参阅[资源关联 \(概述\)](#)。有关有效字符的详细信息, 请参阅[命名约定](#)。
- **keyItem** 是必需的, 并且只能是在同一记录中唯一的结构字段。必须使用未限定引用来指定键字段; 例如, 使用 `myItem` 而不是 `myRecord.myItem`。(然而, 在 EGL 语句中, 可以象引用任何字段一样引用键字段。)

另请参阅[支持变长记录的属性](#)。

mqRecord

MQ 记录是一个固定记录, 它允许您访问 MQSeries 消息队列。有关详细信息, 请参阅[MQSeries 支持](#)。

relativeRecord

相对记录是一个固定记录, 它允许您使用其记录具有下列属性的数据集:

- 固定长度

- 可以通过一个表示文件中记录的顺序位置的整数来访问

类型为 `relativeRecord` 的部件的属性如下所示:

- **fileName** 是必需的。有关输入的含义的详细信息，请参阅[资源关联（概述）](#)。有关有效字符的详细信息，请参阅[命名约定](#)。
- **keyItem** 是必需的。键字段可以是下列任何内存区：
 - 同一记录中的结构字段
 - 记录中对于程序来说是全局的或对于访问该记录的函数来说是局部的结构字段
 - 对于程序来说是全局的，对于访问该记录的函数来说是局部的基本变量

必须使用未限定引用来指定键字段。例如，使用 `myItem` 而不是 `myRecord.myItem`。（在 EGL 语句中，可以象引用任何字段一样引用键字段。）键字段在访问记录的函数的局部作用域中必须是唯一的，或者，必须是不在局部作用域中并在全局作用域中唯一的。

键字段具有下列特征:

- 具有基本类型 BIN、DECIMAL、INT 或 NUM
- 不包含小数位
- 最多允许 9 位数

只有 **get** 和 **add** 语句使用键字段，但键字段必须可以供任何使用记录来进行文件访问的函数使用。

serialRecord

串行记录是一个固定记录，它允许您访问顺序访问的文件或数据集。可以通过调用 **get** 语句来从文件中读取记录，并且一系列 **get next** 语句将按顺序从第一条记录到最后一条记录读取文件记录。可以通过调用 **add** 语句来写入文件，该语句将新记录放在文件末尾。

串行记录属性包括 **fileName**，此属性是必需的。有关该属性的输入的含义的详细信息，请参阅[资源关联（概述）](#)。有关有效字符的详细信息，请参阅[命名约定](#)。

另请参阅[支持变长记录的属性](#)。

sqlRecord

SQL 记录是这样—个记录（或固定记录），它在您访问关系数据库时提供特殊服务。

在缺省情况下，该部件是一个记录部件，但如果字段定义前有级别号，则该部件是一个固定记录部件。

每个部件具有下列属性:

- **tableNames** 中的一个条目标识与该部件相关联的 SQL 表。在一个连接中可以引用多个表，但是进行了一些限制以确保不使用单个 EGL 语句来写入多个表。可以将给定的表名与标号相关联，后者是用来在 SQL 语句中引用表的可选短名称。
- **defaultSelectCondition** 是可选的。这些条件成为缺省 SQL 语句中的 WHERE 子句的一部分。在 EGL **open** 或 **get** 语句或像 **get next** 或 **get previous** 这样的语句中使用 SQL 记录时，WHERE 子句是很有用的。

在大多数情况下，SQL 缺省选择条件都将补充另一个条件，该条件基于 SQL 记录中的键字段值与 SQL 表的键列之间的关联。

- **tableNameVariables** 是可选的。您可以指定一个或多个变量，在运行时，这些变量的内容确定要访问的数据库表，如动态 SQL 中所述。
- **keyItems** 是可选的。每个键字段都只能是在同一记录中唯一的结构字段。必须使用未限定引用来指定那些字段中的每一个；例如，使用 *myItem* 而不是 *myRecord.myItem*。（然而，在 EGL 语句中，可以象引用任何字段一样引用键字段。）

有关详细信息，请参阅 *SQL 支持*。

相关概念

- 第 123 页的『固定记录部件』
- 第 219 页的『动态 SQL』
- 第 242 页的『MQSeries 支持』
- 第 122 页的『记录部件』
- 第 277 页的『资源关联和文件类型』
- 第 209 页的『SQL 支持』

相关参考

- 第 511 页的『add』
- 第 517 页的『close』

- 第 430 页的『数据初始化』
- 第 520 页的『delete』
- 第 522 页的『execute』
- 第 533 页的『get』
- 第 544 页的『get next』
- 第 549 页的『get previous』
- 第 606 页的『MQ 记录属性』
- 第 612 页的『命名约定』
- 第 561 页的『open』
- 第 574 页的『prepare』
- 第 673 页的『支持变长记录的属性』
- 第 673 页的『记录和文件类型交叉引用』
- 第 576 页的『replace』
- 第 61 页的『SQL 项属性』
- 第 860 页的『terminalID』

创建 EGL 程序部件

EGL 程序部件是用来生成 Java 程序、Java 包装器或 EJB 会话 bean 的主逻辑单元。有关更多信息，请参阅 *程序部件*。

当您在工作台中创建文件时，程序部件将自动添加到程序文件中，并适当地命名。程序文件规范只允许每个文件只有一个程序部件，并且要求程序名与文件名相匹配。

要使用程序部件来创建程序文件，执行下列操作：

1. 标识用来包含文件的项目或文件夹。如果还没有项目或文件夹，则必须创建项目或文件夹。

2. 在工作台中，单击文件 > 新建 > 程序。
3. 选择将包含 EGL 文件的项目或文件夹，然后选择包。由于程序名将与文件名完全相同，因此选择一个遵从 EGL 部件名约定的文件名。在“EGL 源文件名”字段中，输入 EGL 文件的名称，例如，myEGLprg。选择 EGL 程序类型（有关详细信息，请参阅 *EGL 源格式的基本程序* 或 *EGL 源格式的 TextUI 程序*）。如果程序部件是主程序，则进行单击以除去“作为被调用程序创建”的选取标记。
4. 单击完成按钮。

相关概念

第 13 页的『EGL 项目、包和文件』

第 1 页的『EGL 简介』

『程序部件』

相关任务

第 117 页的『创建 EGL 源文件夹』

相关参考

第 666 页的『EGL 源格式的基本程序』

第 118 页的『创建 EGL 源文件』

第 612 页的『命名约定』

第 668 页的『EGL 源格式的文本用户界面程序』

程序部件

程序部件定义运行时 Java 程序中的中央逻辑单元。有关主程序、被调用程序以及程序类型（基本和 textUI）的概述，请参阅部件。

任何类型的程序部件都可以包含名为 *main* 的函数，该函数表示在程序启动时运行的逻辑。程序可以包含其它函数，并可以访问位于该程序外部的函数。*main* 函数可以调用其它那些函数，并且任何函数都可以将控制权传递给其它程序。

最重要的程序属性如下所示：

- 每个参数都引用一个内存区，该内存区包含从调用程序接收的数据。参数是程序全局的，并且只在被调用程序中有效。
- 每个变量都引用一个内存区，该内存区是在程序中分配的，并且是程序全局的。
- 表单组是表单的集合，那些表单将数据提供给用户：
 - 基本程序可以通过打印表单将数据提供给打印机
 - 文本程序可以（通过文本表单）交互地显示数据，也可以将数据提供给打印机

有关详细信息，请参阅 *FormGroup* 部件。

- 输入记录是一个全局内存区，当从另一个程序以异步方式转移控制权时，此内存区接收数据。输入记录仅在主程序中才可用。
- 在文本主程序中，*segmented* 属性确定程序在发出 **converse** 语句来显示文本表单之前要自动执行的操作。有关详细信息，请参阅分段。
- 并且，在文本程序中，输入表单在程序启动时具有下列两项用途的其中一项：
 - 将表单显示给从监视器或终端调用程序的用户

- 另外，用户输入的数据被接收到输入表单中，后者是程序本身中的内存区。此情况仅适用于延迟程序切换的情况，延迟程序切换是指由 **show** 语句的变体导致的两步骤控制权转移：
 1. 一个程序提交文本表单给用户，然后终止
 2. 用户提交表单，依靠该表单中的信息，提交操作自动调用第二个程序，该程序包含输入表单

要获取程序属性的完整列表，请参阅程序部件属性。

相关概念

- 第 141 页的『FormGroup 部件』
- 第 130 页的『函数部件』
- 第 16 页的『部件』
- 第 53 页的『引用 EGL 中的变量』
- 第 147 页的『文本应用程序中的分段』

相关任务

- 第 127 页的『创建 EGL 程序部件』

相关参考

- 第 441 页的『EGL 中的内容辅助』
- 第 430 页的『数据初始化』
- 第 448 页的『EGL 源格式』
- 第 80 页的『EGL 语句』
- 第 664 页的『EGL 源格式的程序部件』
- 第 670 页的『程序部件属性』

创建 EGL 函数部件

函数部件是一个逻辑单元，它或者包含程序中最先执行的代码，或者是从另一个函数中调用的。EGL 函数部件包含在 EGL 文件中。要创建 EGL 函数部件，执行下列操作：

1. 标识要包含函数部件的 EGL 文件并在 EGL 编辑器中打开该文件。如果还没有 EGL 文件，则必须创建它。
2. 根据 EGL 语法输入函数部件的细节（有关详细信息，请参阅 *EGL 源代码格式的函数部件*）。可使用内容辅助以将函数部件语法的概要内容放在文件中。
3. 保存 EGL 文件。

相关概念

- 第 13 页的『EGL 项目、包和文件』
- 第 130 页的『函数部件』

相关任务

- 第 118 页的『创建 EGL 源文件』
- 第 118 页的『将 EGL 模板与内容辅助配合使用』

相关参考

- 第 441 页的『EGL 中的内容辅助』

- 第 473 页的『函数调用』
- 第 481 页的『EGL 源格式的函数部件』
- 第 612 页的『命名约定』

函数部件

函数部件是一个逻辑单元，它或者包含程序中最先执行的代码，或者是从另一个函数中调用的。包含程序中最先执行的代码的函数名为 *main*。

函数部件可以包含下列属性：

- 返回值，它描述函数部件返回给调用程序的数据
- 一组参数，每个这样的参数都引用由另一个逻辑部件分配并传递的内存
- 一组其它变量，每个这样的变量都分配作为该函数的局部内存的其它内存
- EGL 语句
- 关于函数是否需要程序上下文的说明，在 *containerContextDependent* 中作了描述

main 函数由于不能返回值或包含参数而与众不同，它必须在程序部件内进行声明。

相关概念

- 第 16 页的『部件』
- 第 128 页的『程序部件』
- 第 53 页的『引用 EGL 中的变量』
- 第 209 页的『SQL 支持』

相关参考

- 第 425 页的『*containerContextDependent*』
- 第 430 页的『数据初始化』
- 第 448 页的『EGL 源格式』
- 第 473 页的『函数调用』
- 第 481 页的『EGL 源格式的函数部件』
- 第 80 页的『EGL 语句』

创建 EGL 库部件

EGL 库部件包含一组可以由程序、PageHandler 或其它库使用的函数、变量和常量。要创建 EGL 库部件，执行下列操作：

1. 标识用来包含文件的项目或文件夹。如果还没有项目或文件夹，则必须创建项目或文件夹。
2. 在工作台中，单击文件 > 新建 > 库。
3. 选择将包含 EGL 文件的项目或文件夹，然后选择包。由于库名将与文件名相同，因此选择一个符合 EGL 部件名约定的文件名。在“EGL 源文件名”字段中，输入 EGL 文件的名称，例如，myLibrary。
4. 单击下列其中一个单选按钮来选择库的类型：
 - 基本 - 创建基本库
 - 本机 - 创建本机库
5. 单击完成按钮。

相关概念

第 13 页的『EGL 项目、包和文件』

第 1 页的『EGL 简介』

『类型为 `basicLibrary` 的库部件』

『类型为 `basicLibrary` 的库部件』

相关任务

第 117 页的『创建 EGL 源文件夹』

第 116 页的『创建 EGL Web 项目』

相关参考

第 118 页的『创建 EGL 源文件』

第 592 页的『EGL 源格式的库部件』

第 612 页的『命名约定』

类型为 `basicLibrary` 的库部件

类型为 `basicLibrary` 的库部件包含一组可以由程序、`PageHandler` 或其它库使用的函数、变量和常量。建议您通过使用库来最大程度地重用公共代码和值。

类型说明 `basicLibrary` 指示该部件将生成到可编译单元中并且包括用于本地执行的 EGL 值和代码。如果未指定关键字 `type`，则此类型为缺省值。有关创建库以从 EGL 生成的 Java 程序中访问本地 DLL 的详细信息，请参阅类型为 `nativeLibrary` 的库部件。

类型为 `basicLibrary` 的库的规则如下所示：

- 可以在不指定库名的情况下引用库的函数、变量和常量，但是仅当将该库包括在特定于程序的使用声明中时才能这样做。
- 库函数可以访问任何与调用程序或 `PageHandler` 相关联的系统变量。下列规则适用：
 - 库中的函数接收到作为自变量的记录时，该记录不能用于输入或输出（I/O）或测试 I/O 状态（如 `endOfFile`）。但是，调用该库的代码可以任一方式使用该记录。
 - 在库中声明记录时，基于库的函数可将该记录用于输入或输出（I/O）或测试 I/O 状态（如文件结束）。但是，调用该库的代码不能以任一方式使用该记录。
- 库函数可以使用除下列语句以外的任何语句：
 - `converse`
 - `forward`
 - `show`
 - `transfer`
- 库不能访问文本表单。
- 访问打印表单的库必须包括相关表单组的使用声明。
- 可以对函数、变量或常量声明使用修饰符 `private`，以防止在库外部使用该元素。
- 被声明为公用函数（这是缺省情况）的库函数在库外部可用，并且不能带有具有松散类型的参数。松散类型是一种特殊的基本类型，仅当您希望参数接受特定范围的自变量长度时，才能使用松散类型。有关松散类型的详细信息，请参阅 *EGL 源格式的函数部件*。

库是独立于使用它的部件生成的。EGL 运行时通过使用库属性 `alias` 的设置（缺省为 EGL 库名）来访问库部件。

在运行时，该库是在第一次使用时装入的，并且在访问该库的程序或 PageHandler 离开内存（在运行单元结束时）时卸装的。

每当装入 PageHandler 时，该 PageHandler 都将获得库的新副本。并且，对于被另一个库调用的库，只要调用库存在于内存中，被调用库也一直存在于内存中。

由于常量在引用它们的程序和 PageHandler 中是作为文字生成的，所以，在运行时不装入只包含常量的库。

相关概念

- 第 531 页的『forward』
- 第 481 页的『EGL 源格式的函数部件』
- 第 592 页的『EGL 源格式的库部件』
- 第 131 页的『类型为 basicLibrary 的库部件』
- 第 678 页的『运行单元』
- 第 147 页的『文本应用程序中的分段』
- 第 588 页的『show』
- 第 589 页的『transfer』
- 第 875 页的『使用声明』

相关参考

- 第 519 页的『converse』
- 第 531 页的『forward』
- 第 481 页的『EGL 源格式的函数部件』
- 第 592 页的『EGL 源格式的库部件』
- 第 678 页的『运行单元』
- 第 147 页的『文本应用程序中的分段』
- 第 588 页的『show』
- 第 589 页的『transfer』
- 第 875 页的『使用声明』

类型为 nativeLibrary 的库部件

类型为 nativeLibrary 的库允许 EGL 生成的 Java 代码调用单个在本地运行的 DLL。该 DLL 的代码不是用 EGL 语言编写的。有关开发基本库（包含用 EGL 语言编写的共享函数和值）的信息，请参阅类型为 basicLibrary 的库部件。

在类型为 nativeLibrary 的库中，每个函数的用途在于为 DLL 函数提供接口。不能在 EGL 函数中定义语句，也不能在库中的任何位置声明变量或常量。

EGL 运行时通过使用 EGL 函数属性 **alias** 的设置（缺省为 EGL 函数名）来访问基于 DLL 的函数。如果基于 DLL 的函数的名称不符合命名约定中描述的约定，则明确设置该属性。

库属性 **callingConvention** 指定 EGL 运行时如何在下列两种代码之间传递数据：

- 调用库函数的 EGL 代码；以及
- DLL 中的函数。

现在对 **callingConvention** 可用的唯一值是 *14GL*：

- 数据是以符合 Informix 堆栈格式的方式传递的。每个输入参数放在输入堆栈上，而每个输出参数放在输出堆栈上。
- 不能传递记录或字典之类的自变量。仅下列各项有效：
 - 基本变量，包括类型为 ANY 的变量
 - dataTables、打印表单、文本表单和固定记录中的字段（仅当字段没有子结构时）

库函数中的参数必须是基本变量，并且可以是 ANY 类型，但不能是松散类型，也不能包括 **field** 修饰符。

库属性 **dllName** 指定 DLL 名称，这是最终名称；在部署时不能覆盖它。如果未对库属性 **dllName** 指定值，则必须在 Java 运行时属性 `vgj.defaultI4GLNativeLibrary` 中指定 DLL 名称。因为只有一个这样的 Java 运行时属性对运行单元可用，所以除了在 EGL 库中标识的 DLL 之外，只能指定一个 DLL。

不管您是在开发时（在 **dllName** 中）还是在部署时（在 `vgj.defaultI4GLNativeLibrary`）指定 DLL 名称，该 DLL 必须驻留在运行时变量中标识的目录路径中；该变量为 `PATH`（在 Windows 2000/NT/XP 上）或 `LIBPATH`（在 UNIX 平台上）。

库函数将自动声明为公用函数以确保它们在库外部可用。在其它 EGL 代码中，可以在不指定库名的情况下引用函数（仅使用其函数别名），但仅当将该库包括在特定于程序的使用声明中时才能这样做。

EGL 库被生成为 Java 类，该类与访问该库的代码和 DLL 是分开的。EGL 运行时通过使用库属性 **alias** 的设置（缺省为 EGL 库名）来访问该类。如果库部件的名称不符合 Java 约定，则明确设置该属性。

在运行时，当第一次使用 DLL 时会将其装入，并在访问该程序或 PageHandler 离开内存（在运行单元结束时）时卸装该库。

每当装入 PageHandler 时，该 PageHandler 都将获得 DLL 的新副本。并且，对于被类型为 `basicLibrary` 的 EGL 库调用的 DLL，只要调用库在内存中，被调用的 DLL 也会在内存中。

以下本地库提供对用 C 编写的 DLL 的访问：

```
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
```

相关概念

- 第 315 页的『Java 运行时属性』
- 第 131 页的『类型为 basicLibrary 的库部件』

相关参考

- 第 481 页的『EGL 源格式的函数部件』
- 第 493 页的『Java 运行时属性（详细信息）』
- 第 592 页的『EGL 源格式的库部件』
- 第 612 页的『命名约定』
- 第 678 页的『运行单元』
- 第 875 页的『使用声明』

创建 EGL dataTable 部件

EGL dataTable 部件使数据结构与结构的初始值数组相关联。要创建 EGL dataTable 部件，执行下列操作：

1. 标识用来包含文件的项目或文件夹。如果还没有项目或文件夹，则必须创建项目或文件夹。
2. 在工作台中，单击**文件 > 新建 > 数据表**。
3. 选择将包含 EGL 文件的项目或文件夹，然后选择包。由于 dataTable 名将与文件名相同，因此选择一个符合 EGL 部件名约定的文件名。在“EGL 源文件名”字段中，输入 EGL 文件的名称，例如，myDataTable。选择 dataTable 子类型（有关详细信息，请参阅 *EGL 源格式的数据表部件*）。
4. 单击**完成**按钮。

相关概念

- 『DataTable』
- 第 13 页的『EGL 项目、包和文件』
- 第 1 页的『EGL 简介』

相关任务

- 第 117 页的『创建 EGL 源文件夹』
- 第 116 页的『创建 EGL Web 项目』

相关参考

- 第 118 页的『创建 EGL 源文件』
- 第 432 页的『EGL 源格式的 DataTable 部件』
- 第 612 页的『命名约定』

DataTable

EGL dataTable 主要由下列组件组成：

- 一个结构，每个顶级项都定义一列。
- 与那些列一致的一组值。该组值的每个元素都定义一行。

例如，错误消息的 dataTable 可能包括下列组件：

- 一个数字字段和一个字符字段的声明
- 类似于以下的配对值列表：

```
001 Error 1
002 Error 2
003 Error 3
```

不必像声明记录或数据项那样声明 `dataTable`。而是可访问 `dataTable` 的任何代码都会将该部件视为变量。有关部件访问的详细信息，请参阅[对部件的引用](#)。

任何可访问 `dataTable` 的代码都可以选择在使用声明中引用部件名。

dataTable 的类型

某些类型的 `dataTable` 是用于运行时验证的；特别是用于存放要与表单输入进行比较的数据。（在声明表单部件时，让 `dataTable` 与输入字段相关。）有三种验证类型的 `dataTable`：

matchValidTable

用户的输入必须与第一个 `dataTable` 列中的值相匹配。

matchInvalidTable

用户的输入必须与第一个 `dataTable` 列中的任何值不同。

rangeChkTable

用户的输入必须与某个值相匹配，该值至少介于一个 `dataTable` 行的第一列和第二列的值之间。（范围是包括边界的；当用户的输入与任何行的第一列或第二列的值相匹配时，输入有效。）

其它类型的 `dataTable` 如下所示：

msgTable

包含运行时消息。

basicTable

包含程序逻辑所使用的其它信息；例如，国家或地区以及相关代码的列表。

生成 DataTable

`dataTable` 生成操作的输出是一对文件，这两个文件都具有 `dataTable` 的名称。一个文件具有扩展名 `.java`，另一个文件具有扩展名 `.tab`。`.tab` 文件不是由 Java 编译器处理的，但是它位于包含包的目录结构的根目录中。例如，如果包是 `my.product.package`，目录结构是 `my/product/package`，则 `.tab` 文件位于包含子目录 `my` 的目录中。

如果在一个包中已经生成了 `dataTable`，则不需要再在该包中生成同样的 `dataTable`。

不需要生成 `dataTable` 时，为了节省生成时间，请对构建描述符选项 `genTables` 指定 `NO`。

dataTable 的属性

可以设置下列属性：

- **alias** 包含在生成的输出的名称中。如果未指定别名，则会使用部件名。
- **shared** 属性指示 `dataTable` 是否可以由多个用户访问。缺省值为 `no`。
- **resident** 属性指示即使没有任何程序使用 `dataTable`，是否也将该 `dataTable` 保留在内存中。（程序在第一次被访问时进入内存。）缺省值为 `no`。仅当共享说明也是 `yes` 时才能指定 `yes`。

相关概念

第 20 页的『对部件的引用』

相关参考

第 432 页的『EGL 源格式的 DataTable 部件』

第 875 页的『使用声明』

将代码段插入到 EGL 和 JSP 文件中

“片段”视图允许您将可重复使用的编程对象插入到代码中。“片段”视图包含若干 EGL 代码块以及许多其它技术的代码。可使用提供的代码段或将您自己的代码段添加至“片段”视图。有关使用“片段”视图的更多信息，请参阅“片段”视图。

要将 EGL 代码段插入到代码中，执行下列操作：

1. 打开要对其添加片段的文件。
2. 打开“片段”视图。
 - a. 单击窗口 > 显示视图 > 其它。
 - b. 展开基本并单击片段。
 - c. 单击确定。
3. 在“片段”视图中，展开 **EGL** 抽屉。此抽屉包含可用的 EGL 代码段。
4. 使用下列其中一个方法将片段插入到文件中：
 - 单击片段并将其拖到源代码中。
 - 双击片段以在当前光标位置插入该片段。将会显示一个窗口，其中描述片段中的变量和值。如果这样的话，单击**插入**。

注：如果光标变成中间有东西穿过的圆圈，指示不能在该位置插入片段，则表示您可能会尝试将该片段插入到错误的位置。检查片段的详细信息以了解它应该在插入在代码中的什么位置。

5. 根据代码的情况，对片段中的函数、变量和数据部件的预定义名称作适当的更改。大多数片段包括说明哪些名称需要更改的注释。

下面是 EGL 中可用的片段：

表 6. EGL 中可用的片段

片段名	描述
setCursorFocus	一个 JavaScript™ 函数，用于将光标焦点放在 Web 页面上的指定表单字段中。
autoRedirect	一个 JavaScript 函数，用于测试会话变量是否存在。如果会话变量不存在，它会将浏览器转至另一页面。
getClickedRowValue	一个 EGL 函数，用于检索数据表中的被单击行的超链接值。
databaseUpdate	一个 EGL 函数，用于在从 PageHandler 传递记录时更新关系表的一行。

相关概念

“片段”视图

相关任务

第 118 页的『将 EGL 模板与内容辅助配合使用』

第 138 页的『将焦点设置为表单字段』

『针对会话变量测试浏览器』

第 139 页的『检索数据表中被单击的行的值』

第 139 页的『更新关系表中的行』

相关参考

将焦点设置为表单字段

“片段”视图的 JSP 抽屉中的 `setCursorFocus` 片段是一个 JavaScript 函数，它会将光标焦点设置为 Web 页面上的指定表单字段。它必须放置在 JSP 页面中的 `<script>` 标记中。要插入并配置此片段，遵循下列指示信息：

1. 将片段代码插入到页面的源代码中。有关更多信息，请参阅插入 *EGL 代码段*。
2. 将 `[n]` 替换为接收焦点的表单字段的编号。例如，使用 `[3]` 以将焦点设置为页面上的第 4 个字段。
3. 将表单名设置为 `form1`。
4. 将 JSP 页面的 `<body>` 标记更改为 `<body onload="setfocus();">`。

此片段插入的代码如下所示：

```
function setFocus() {  
    document.getElementById('form1').elements[n].select();  
    document.getElementById('form1').elements[n].focus();  
}
```

相关任务

第 137 页的『将代码段插入到 EGL 和 JSP 文件中』

针对会话变量测试浏览器

“片段”视图的 JSP 抽屉中的 `autoRedirect` 片段测试会话变量是否存在。如果会话变量不存在，它会将浏览器转至另一页面。此片段必须放在 JSP 页面的 `<head>` 标记中 `<pageEncoding>` 标记的后面。要插入并配置此片段，遵循下列指示信息：

1. 将片段代码插入到页面的 `<head>` 标记中 `<pageEncoding>` 标记的后面。有关更多信息，请参阅插入 *EGL 代码段*。
2. 将 `{SessionAttribute}` 替换为要测试的会话变量的名称。
3. 将 `{ApplicationName}` 替换为项目或应用程序的名称。
4. 如果缺少会话变量，将 `{PageName}` 替换为浏览器将重定向至的页面的名称。

此片段插入的代码如下所示：

```
<%  
if ((session.getAttribute("userID") == null ))  
{  
    String redirectURL =  
        "http://localhost:9080/EGLWeb/faces/Login.jsp";  
    response.sendRedirect(redirectURL);  
}  
%>
```

相关任务

第 137 页的『将代码段插入到 EGL 和 JSP 文件中』

检索数据表中被单击的行的值

“片段”视图的 EGL 抽屉中的 `getClickedRowValue` 片段是一个函数，用于检索数据表中的被单击行的超链接值。此片段必须放在 EGL `PageHandler` 中。此片段具有下列先决条件：

1. JSP 页面具有数据表。
2. JSP 标识的名称未更改，仍是缺省值。
3. 该页面在 `faces-config.xml` 的作用域中被定义为请求，而不是会话。

要插入并配置此片段，遵循下列指示信息：

1. 将片段的代码插入到 `PageHandler` 中。有关更多信息，请参阅插入 *EGL 代码段*。
2. 定义用于接收被单击值的字符型或字符串变量。
3. 将命令超链接（从“选用板”视图中的“Faces 组件”抽屉）添加至数据表中的字段。
4. 对于命令超链接的目标，指定 JSP 页面的名称。超链接链接至它自己的页面。
5. 将参数添加至超链接并对参数指定名称，该名称就是 `PageHandler` 中接收被单击值的变量的名称。
6. 将 `action` 属性（位于“属性”视图的“全部”选项卡上）设置为 `getVal()` 函数。

此片段插入的代码如下所示：

```
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
```

相关任务

第 137 页的『将代码段插入到 EGL 和 JSP 文件中』

更新关系表中的行

“片段”视图的 EGL 抽屉中的 `databaseUpdate` 片段是一个 EGL 函数，用于在从 `PageHandler` 传递记录时更新关系表的一行。此片段将放在 EGL 库中。要插入并配置此片段，遵循下列指示信息：

1. 将片段的代码插入到 `PageHandler` 中。有关更多信息，请参阅插入 *EGL 代码段*。
2. 将 `{tableName}` 和 `{keyColumn}` 替换为表名及其主键列。

此片段插入的代码如下所示：

```
Function updateRec(${TableName}New ${TableName})

  // Function name - call this function
  // passing the ${TableName} Record as a parameter
  ${TableName}Old ${TableName};

  // A copy of the Record, used
```

```

// to lock the table row and to obtain
// the existing row values prior to update try
${TableName}Old.${KeyColumn} =
    ${TableName}New.${KeyColumn};
get ${TableName}Old forUpdate;

// Get the existing row.
// Note that if you had custom processing to do,
// you would insert your code after this call
move ${TableName}New to ${TableName}Old byName;

//Move the updated values to the copy-row
replace ${TableName}Old;

//And replace the row in the database.
sysLib.commit();

//Commit your changes to the Database
onException
    //If the update fails...
    sysLib.rollback();
    // cancel all database updates
    // (assuming this is permitted
    // by your database) and call
    // a custom error handling routine
end
end

```

相关任务

第 137 页的『将代码段插入到 EGL 和 JSP 文件中』

使用文本和打印表单

创建 EGL formGroup 部件

EGL formGroup 部件定义文本和打印表单的集合。要创建 EGL formGroup 部件，执行下列操作：

1. 标识用来包含文件的项目或文件夹。如果还没有项目或文件夹，则必须创建项目或文件夹。
2. 在工作台中，单击**文件 > 新建 > 表单组**。
3. 选择将包含 EGL 文件的项目或文件夹，然后选择包。由于 formGroup 名将与文件名相同，因此选择一个符合 EGL 部件名约定的文件名。在“EGL 源文件名”字段中，输入 EGL 文件的名称，例如，myFormGroup。
4. 单击**完成**按钮。

相关概念

第 13 页的『EGL 项目、包和文件』

第 150 页的『EGL 表单编辑器概述』

第 151 页的『使用 EGL 表单编辑器编辑表单组』

『FormGroup 部件』

第 1 页的『EGL 简介』

相关任务

第 117 页的『创建 EGL 源文件夹』

第 116 页的『创建 EGL Web 项目』

相关参考

第 118 页的『创建 EGL 源文件』

第 467 页的『EGL 源格式的表单部件』

第 612 页的『命名约定』

FormGroup 部件

EGL FormGroup 部件有两项用途：

- 定义文本和打印表单的集合。（部件特有的表单是在该部件中定义的，或者是通过使用声明包括的。若干个 FormGroup 部件的公共表单是通过使用声明包括的。）
- 不定义或定义多个**浮动区域**，如**表单部件**中所述

不必声明表单组，而象是正在声明记录或数据项一样。或者，仅当出现下列情况时程序才访问 FormGroup 部件（及相关表单）：

- 程序可访问 FormGroup 部件的位置，如**对部件的引用**中所述
- 程序中的使用声明引用 FormGroup 部件

程序不能包含两个以上的 formGroup 部件；如果指定了两个，则其中一个必须是**帮助组**。帮助组包含一个或多个**帮助表单**，后者是只读的表单，它们提供信息以对用户击键作出响应。

在运行时，仅当生成 FormGroup 时表单才可用。生成的输出是 FormGroup 部件的类和每个表单部件的类。

在准备时，每一个实体都被处理成独立的运行时装入模块。EGL 运行时处理生成的程序与特定于表单的代码进行的交互。

不能单独地生成表单部件。

相关概念

『表单部件』

第 20 页的『对部件的引用』

第 151 页的『使用 EGL 表单编辑器编辑表单组』

相关参考

第 875 页的『使用声明』

表单部件

表单部件是显示单元。它描述同时向用户显示的一组字段的布局 and 特征。

不必声明表单，而象是正在声明记录或数据项一样。要访问表单部件，程序必须包含引用相关表单组的使用声明。

表单部件为两种类型的其中一种类型，即文本表单或打印表单：

- 类型为文本的表单定义命令窗口中显示的布局。除了一个例外情况以外，任何文本表单都可以具有常量字段和变量字段，包括接受用户输入的变量字段。该例外情况是帮助表单，它仅仅用于显示常量信息。
- 打印类型的表单定义发送至打印机的布局。任何打印表单都可以同时具有常量和变量字段。

表单属性确定输出在屏幕上或页上的大小和位置，并指定该输出的格式特征。

给定的表单可以在一个或多个设备上显示，每个设备都是输出外围设备，或者是等同于输出外围设备的操作设备：

- 屏幕设备是终端、监视器或终端仿真器。输出表面是屏幕。
- 打印设备是可以发送至打印机的文件，或者是打印机本身。输出表面是页。

无论是文本类型还是打印类型，都可以对表单进行进一步分类，如下所示：

- 固定表单具有与设备输出表面相关的特定开始行和列。例如，可以指定固定的打印表单，以从一页上的第 10 行第 1 列开始。
- 浮动表单没有特定的开始行或列；或者，浮动表单的位置是所声明的输出表面子区域中的下一个空行。声明的子区域被称为浮动区域。

可以将浮动区域声明为从第 10 行开始并扩展至第 20 行的矩形，它具有输出设备的最大宽度。如果有一个具有相同宽度的一行浮动表单，则可以构造一个循环，该循环将下列操作执行 20 次：

1. 将数据放入浮动映射中
2. 将浮动映射写入浮动区域的下一行

在 `FormGroup` 部件中声明了一个或多个浮动区域，但只有一个浮动区域能接受特定设备的浮动表单。如果尝试在没有浮动区域的情况下显示浮动表单，则会将整个输出表面看作浮动区域。

- 部分表单小于特定设备的输出表面的标准大小。可以声明和定位部分表单，以便将多个表单显示于不同的水平位置。虽然可以指定部分表单的开始和结束列，但是无法显示相邻的表单。

特定于表单类型的其它详细信息:

- 打印表单
- 文本表单

相关概念

第 144 页的『打印表单』

第 145 页的『文本表单』

第 151 页的『使用 EGL 表单编辑器编辑表单组』

第 156 页的『EGL 表单编辑器中的表单模板』

相关任务

第 152 页的『在 EGL 表单编辑器中创建表单』

相关参考

第 464 页的『EGL 源格式的 `FormGroup` 部件』

第 467 页的『EGL 源格式的表单部件』

创建 EGL 打印表单

打印表单是定义要发送至打印机的布局的 EGL 表单部件。要创建 EGL 打印表单，执行下列操作:

1. 标识要包含打印表单的 EGL 文件并在 EGL 编辑器中打开该文件。如果还没有 EGL 文件，则必须创建它。
2. 根据 EGL 语法输入打印表单的细节（有关详细信息，请参阅 *EGL 源代码格式的表单部件*）。可使用内容辅助以将表单部件语法的大纲放在文件中。
3. 保存 EGL 文件。

相关概念

第 13 页的『EGL 项目、包和文件』

第 142 页的『表单部件』

第 144 页的『打印表单』

第 151 页的『使用 EGL 表单编辑器编辑表单组』

相关任务

第 118 页的『创建 EGL 源文件』

第 118 页的『将 EGL 模板与内容辅助配合使用』

第 152 页的『在 EGL 表单编辑器中创建表单』

相关参考

第 441 页的『EGL 中的内容辅助』

第 467 页的『EGL 源格式的表单部件』

第 612 页的『命名约定』

打印表单

在表单部件中引入了表单及其类型。当前页概述如何显示打印表单。

打印过程: 打印是一个两步骤过程:

- 首先, 您编写 **print** 语句, 每个语句都向运行时缓冲区中添加一个表单
- 接着, EGL 运行时添加开始新页所需的符号、将所有缓存表单发送至打印设备并擦除缓冲区内容。那些服务是作为对下列任何情况的响应而提供的:
 - 程序对去往同一打印设备的打印表单运行 **close** 语句; 或者
 - 程序处于分段方式 (如分段中所述) 并运行 **converse** 语句; 或者
 - 程序已被非 EGL 程序 (并且是非 VisualAge Generator 程序) 调用, 并且被调用程序结束; 或者
 - 运行单元中的主程序结束。

对于多表单输出, 必须按照所需的表单显示顺序来调用 **print** 语句。请参照以下示例:

- 在输出的顶部, 一个固定表单标识了采购公司和订单号
- 在后续的浮动区域中, 一系列格式完全相同的浮动表单标识了公司订单中的每件商品
- 在输出的底部, 一个固定表单指示了浏览整个商品列表所需的屏幕数或页数

可以通过提交一系列全都对打印表单执行操作的 **print** 语句来得到该输出。那些语句按照以下顺序引用表单:

1. 顶部表单
2. 浮动表单, 由一个循环中重复调用的 **print** 语句显示
3. 底部表单

在各种情况下, 会插入开始新页所需的符号, 但在发出 **print** 语句之前, 可以通过调用系统函数 `ConverseLib.pageEject` 来插入那些符号。

固定表单注意事项: 下列情况适用于固定表单:

- 如果对一个固定表单发出 **print** 语句, 而该表单的开始行大于当前行, 则 EGL 将插入使打印设备前进至指定行所需的符号。同样, 如果对一个固定表单发出 **print** 语句, 而该表单的开始行小于当前行, 则 EGL 将插入开始新页所需的符号。
- 如果一个固定表单覆盖了另一固定表单中的某些而非全部行, 则 EGL 自动插入开始新页所需的符号并将第二个固定表单放置在新页上。
- 如果一个固定表单覆盖了另一固定表单的所有行, 则 EGL 将替换现有表单, 而不从缓冲区中清除其余输出。要保留现有输出并将新表单放在下一页上, 请在对新表单发出 **print** 语句之前调用系统函数 `ConverseLib.pageEject`。

浮动表单注意事项: 如果使用浮动表单, 则可能会发生下列错误:

- 您发出 **print** 语句来将浮动表单置于浮动区域末尾之后; 或者
- 您发出使浮动区域至少部分地覆盖固定表单的 **print** 语句, 然后发出 **print** 语句来将浮动表单添加至浮动区域。

在任何一种情况下, 结果都会导致 EGL 插入开始新页所需的符号, 并且浮动表单被置于新页上浮动区域的第一行。例如, 如果该页与上面描述的订单与商品输出类似, 则新页不包含最顶部的固定表单。

打印目标: 当 EGL 处理 **close** 语句以显示打印文件时, 将把输出发送至打印机或数据集。可以在下列三个时间中的任何一个时间指定目标:

- 在测试时 (如 *EGL 调试器* 中所述)
- 在生成时 (如 *资源关联和文件类型* 中所述)
- 在运行时 (如“关于系统变量 `ConverseVar.printerAssociation`”所述)

相关概念

- 第 255 页的『EGL 调试器』
- 第 464 页的『EGL 源格式的 FormGroup 部件』
- 第 467 页的『EGL 源格式的表单部件』
- 第 142 页的『表单部件』
- 第 277 页的『资源关联和文件类型』
- 第 147 页的『文本应用程序中的分段』

相关参考

- 第 722 页的『pageEject()』
- 第 846 页的『printerAssociation』

创建 EGL 文本表单

文本表单是定义要在命令窗口中显示的布局的 EGL 表单部件。要创建 EGL 文本表单, 执行下列操作:

1. 标识要包含文本表单的 EGL 文件并在 EGL 编辑器中打开该文件。如果还没有 EGL 文件, 则必须创建它。
2. 根据 EGL 语法输入文本表单的细节 (有关详细信息, 请参阅 *EGL 源代码格式的表单部件*)。可使用内容辅助以将表单部件语法的大纲放在文件中。
3. 保存 EGL 文件。

相关概念

- 第 13 页的『EGL 项目、包和文件』
- 第 142 页的『表单部件』
- 『文本表单』
- 第 151 页的『使用 EGL 表单编辑器编辑表单组』

相关任务

- 第 118 页的『创建 EGL 源文件』
- 第 152 页的『在 EGL 表单编辑器中创建表单』
- 第 118 页的『将 EGL 模板与内容辅助配合使用』

相关参考

- 第 441 页的『EGL 中的内容辅助』
- 第 467 页的『EGL 源格式的表单部件』
- 第 612 页的『命名约定』

文本表单

在 *表单部件* 中引入了表单及其类型。当前页概述如何显示文本表单。

converse 语句对于允许用户访问单一固定文本表单而言已足够了。仅当用户对显示的表单作出响应之后，程序的逻辑流才会继续。也可以从多个表单构造输出，如下例所示：

- 在输出的顶部，一个固定表单标识了采购公司和订单号
- 在后续的浮动区域中，一系列格式完全相同的浮动表单标识了公司订单中的每件商品
- 在输出的底部，一个固定表单指示了浏览整个商品列表所需的屏幕数

有两个步骤是必需的：

1. 首先，通过编写一系列 **display** 语句来构造订单与商品输出，每一个该语句都将一个表单添加到运行时缓冲区中，但不将数据显示到屏幕上。每个 **display** 语句都对下列其中一个表单执行操作：
 - 顶部表单
 - 浮动表单，由一个循环中重复调用的 **display** 语句显示
 - 底部表单
2. 接着，作为对下列任何一种情况的响应，EGL 运行时将所有缓存文本表单显示到输出设备：
 - 程序运行 **converse** 语句；或者
 - 程序结束。

在大多数情况下，通过编写 **converse** 语句而不是 **display** 语句来显示最后一个屏幕输出表单。

每个固定表单都具有屏幕位置，因此您指定它们的顺序（相互之间的顺序以及相对于重复显示的浮动表单的顺序）无关紧要。将输出发送至屏幕时，将擦除缓冲区的内容。

如果让文本表单相互覆盖，不会发生错误，但下列描述是适用的：

- 如果一个部分表单覆盖了另一固定表单的任何行，则 EGL 将替换现有表单，而不从缓冲区中清除其余输出。在显示新表单之前，如果要擦除现有输出，则在对新表单发出 **display** 或 **converse** 语句之前调用系统函数 `ConverseLib.clearScreen`。
- 如果使用 **display** 或 **converse** 语句来将浮动映射置于浮动区域的底部之后，则将擦除该浮动区域中的所有浮动表单，并且将把添加的表单置于同一浮动区域的第一行。
- 如果浮动表单覆盖固定表单，则下列描述是适用的：
 - 只有浮动区域中的固定表单行才会被浮动表单覆盖
 - 如果固定表单行被包含变量字段的浮动表单行覆盖，则结果是不可预测的

无论是显示一个表单还是显示多个表单，输出目标都是用户在开始运行单元时所用的屏蔽设备。

相关概念

第 142 页的『表单部件』

相关参考

第 467 页的『EGL 源格式的表单部件』

第 464 页的『EGL 源格式的 FormGroup 部件』

第 721 页的『clearScreen()』

文本应用程序中的分段

在发出 **converse** 语句之前，分段关注着程序如何与它的环境进行交互。

缺省情况下，显示文本表单的程序是**非分段的**，这表示该程序的行为就象是始终驻留在内存中并且只为一个用户提供服务一样。在非分段程序发出 **converse** 语句之前，下列规则有效：

- 不落实数据库和其它可恢复的资源
- 不释放锁
- 保留文件和数据库位置
- 不刷新单用户 EGL 表；它们的值在转换之前和之后是相同的
- 同样，不刷新系统变量

被调用程序始终是非分段的。

非分段程序可能更易于编码。例如，在 **converse** 之后，不需要对 SQL 行重新获取锁。缺点包括 SQL 行在用户思考期间被挂起这一事实，这种行为会给需要访问同一 SQL 行的其他用户带来性能问题。

在非分段程序中执行转换之前，可以采用两种技术释放或刷新资源：

- 可以将系统变量 **ConverseVar.commitOnConverse** 设置为 1。在转换之前，结果如下所示：
 - 落实数据库和其它可恢复的资源
 - 释放锁
 - 不保留文件和数据库位置

ConverseVar.commitOnConverse 的设置永远不会影响系统变量或 EGL 表。

- 第二种处理转换的技术是，通过在开发时更改程序属性或通过在运行时将系统变量 **ConverseVar.segmentedMode** 设置为 1，将文本程序的 *segmented* 属性设置为 *yes*。在转换之前，分段将导致下列结果：
 - 落实数据库和其它可恢复的资源
 - 释放锁
 - 不保留文件和数据库位置
 - 刷新单用户 EGL 表；它们的值与程序开始时相同
 - 刷新系统变量；除了一部分跨段保存值的变量以外，变量的值与程序开始时相同

分段程序的行为不受系统变量 **ConverseVar.commitOnConverse** 的值的的影响。

相关概念

第 128 页的『程序部件』

已修正数据标记和 **modified** 属性

文本表单上的每个项都具有已修正数据标记，此标记是一个状态值，它指示是否认为用户自上次显示表单后更改了表单项。

项的已修正数据标记与项的 **modified** 属性不同，后者是在程序中设置的并且预设已修正数据标记的值，如后文所述。

与用户进行交互： 在大多数情况下，当程序向用户显示表单时，已修正数据标记被预设为 *no*；然后，如果用户更改表单项中的数据，则已修正数据标记被设置为 *yes*，并且程序逻辑可以执行下列操作：

- 使用数据表或函数来验证已修改的数据（当项的已修正数据标记是 *yes* 时，此操作自动发生）
- 检测用户是否已修改了项（例如，通过使用 *if item modified* 类型的条件语句）

用户通过在项中输入字符或通过删除字符来设置已修正数据标记。在提交表单之前，已修正数据标记保持已设置状态，即使用户将字段内容恢复为最初显示的值亦如此。

由于错误而重新显示表单时，表单仍处理同一个 *converse* 语句。因此，当重新显示表单时，在执行 *converse* 时被修改的任何字段都将已修正数据标记设置为 *yes*。例如，如果将数据输入到带有验证器函数的字段中，则该函数可以调用 `ConverseLib.validationFailed` 函数来设置错误消息并导致重新显示表单。在这种情况下，由于字段的已修正数据标记仍设置为 *yes*，所以，当操作键被按下时，验证器函数将再次执行。

设置 *modified* 属性： 您可能想让程序在无论用户是否已修改特定字段的情况下都执行某项任务；例如：

- 您可能想强制验证密码字段，即使用户未在该字段中输入数据亦如此
- 可以对关键字段（甚至是对受保护字段）指定验证函数，以便程序始终执行特定交叉字段验证，这表示程序逻辑验证一组字段并考虑一个字段的值是如何影响另一字段的有效性的。

要处理上述情况，可以在程序逻辑中或在表单声明中对特定项设置 **modified** 属性：

- 在表单显示前的逻辑中，包括一个 *set item modified* 类型的语句。结果是，当表单显示时，该项的已修正数据标记已被预设为 *yes*。
- 在表单声明中，将项的 **modified** 属性设置为 *yes*。在这种情况下，下列规则适用：
 - 当表单第一次显示时，该项的已修正数据标记被预设为 *yes*。
 - 如果在表单显示前发生下列任何情况，则表单显示时，已修正数据标记会被预设为 *yes*：
 - 代码运行了 *set item initial* 类型的语句，该语句重新指定项的最初内容和属性值；或者
 - 代码运行了 *set item initialAttributes* 类型的语句，该语句重新指定表单上的每个项的最初属性值（但未指定内容）；或者
 - 代码运行了 *set form initial* 类型的语句，该语句重新指定表单上的每个项的最初内容和属性值；或者
 - 代码运行了 *set form initialAttributes* 类型的语句，该语句重新指定表单上的每个项的最初属性值（但未指定内容）

set 语句影响 **modified** 属性的值，但不影响已修正数据标记的当前设置。*if item modified* 类型的测试基于上次将表单数据返回至程序时正在起作用的已修正数据标记值。如果在逻辑第一次显示表单之前尝试测试项的已修正数据标记，则在运行时会发生错误。

如果需要检测用户（而不是程序）是否已修改了某个项，则确保将该项的已修正数据标记值预设为 *no*：

- 如果在表单声明中将项的 **modified** 属性设置为 *no*，则不要使用 *set item modified* 类型的语句。如果不存在该语句，则会在每次显示表单前将 **modified** 属性自动设置为 *no*。
- 如果在表单声明中将项的 **modified** 属性设置为 *yes*，则在先于表单显示的逻辑中使用 *set item normal* 类型的语句。该语句将 **modified** 属性设置为 *no* 并（作为间接结果）将该项作为未受保护并具有正常强度的项显示。

测试表单是否已被修改： 如果对任何变量表单项将已修正数据标记设置为 *yes*，则将整个表单视为已被修改。如果测试尚未向用户显示的表单的已修改状态，则测试结果是 *FALSE*。

示例： 假定表单 *form01* 具有下列设置：

- 字段 *item01* 的 **modified** 属性设置为 *no*
- 字段 *item02* 的 **modified** 属性设置为 *yes*

以下逻辑显示了各种测试的结果：

```
// tests false because a converse statement
// was not run for the form
if (form01 is modified)
    ;
end

// causes a run-time error because a converse
// statement was not run for the form
if (item01 is modified)
    ;
end

// assume that the user modifies both items
converse form01;

// tests true
if (item01 is modified)
    ;
end

// tests true
if (item02 is modified)
    ;
end

// sets the modified property to no
// at the next converse statement for the form
set item01 initialAttributes;

// sets the modified property to yes
// at the next converse statement for the form
set item02 initialAttributes;

// tests true
// (the previous set statement takes effect only
// at the next converse statement for the form
if (item01 is modified)
```

```

;
end

// assume that the user does not modify either item
converse form01;

// tests false because the program set the modified
// data tag to no, and the user entered no data
if (item01 is modified)
;
end

// tests true because the program set the modified
// data tag to yes
if (item02 is modified)
;
end

// assume that the user does not modify either item
converse form01;

// tests false
if (item01 is modified)
;
end

// tests false because the presentation was not
// the first, and the program did not reset the
// item properties to their initial values
if (item02 is modified)
;
end

```

EGL 表单编辑器概述

EGL 表单编辑器允许您以图形方式编辑 `formGroup` 部件。表单编辑器使用 `formGroup` 部件、它们的表单部件和这些表单部件中的字段的方式与其它图形编辑器使用 Web 页面和 Web 图之类的文件的方式很相似。

表单编辑器具有下列部件:

- 编辑器本身，它显示表单组和该表单组的源代码的图形表示法。可通过单击编辑器底部的设计 and 源代码选项卡来切换显示图形表示法和源代码。对“源代码”视图或“设计”视图的更改将直接反映在另一视图中。
- “属性”视图，它显示当前在编辑器中选择的表单或字段的 EGL 属性。
- “选用板”视图，它显示可在编辑器中创建的表单和字段的类型。
- “大纲”视图，它显示在编辑器中打开的表单组的分层视图。

有关使用表单编辑器的更多信息，请参阅使用 *EGL 表单编辑器编辑表单组*。

相关概念

第 141 页的『FormGroup 部件』

第 142 页的『表单部件』

第 160 页的『EGL 表单编辑器的显示选项』

第 161 页的『EGL 表单编辑器中的表单过滤器』

第 151 页的『使用 EGL 表单编辑器编辑表单组』

相关任务

第 152 页的『在 EGL 表单编辑器中创建表单』

第 160 页的『设置 EGL 表单编辑器的首选项』

第 156 页的『设置 EGL 表单编辑器选用板条目的首选项』

使用 EGL 表单编辑器编辑表单组

EGL 表单编辑器允许您以图形方式编辑 formGroup 部件。表单编辑器使用 formGroup 部件、它们的表单部件和这些表单部件中的字段的方式与其它图形编辑器使用 Web 页面和 Web 图之类的文件的方式很相似。可随时单击编辑器底部的源代码选项卡来查看该编辑器要生成的 EGL 源代码。表单编辑器具有下列功能：

- 表单编辑器可以编辑表单组的大小和属性。要编辑表单组的属性，在表单编辑器中打开该表单组并在“属性”视图中更改它的属性。要调整表单组的大小，在表单编辑器中打开它并从编辑器顶部的列表中选择大小（以字符计）。
- 表单编辑器可以在表单组中创建、编辑和删除表单。要创建表单，在“选用板”视图上单击相应的表单类型，然后在编辑器中绘制表示表单大小和位置的矩形。要编辑表单，单击该表单以选择它，然后使用“属性”视图来编辑它的属性。还可以拖动表单来移动它，或者使用出现在所选表单边框上的调整大小控制柄来调整它的大小。右键单击某个表单以打开其弹出菜单时，将有许多相同选项可用。请参阅在 *EGL 表单编辑器中创建表单*。
- 表单编辑器使用模板来创建常用的表单类型，如弹出表单和弹出菜单。这些表单具有现成的边框、小节和字段。请参阅 *EGL 表单编辑器中的表单模板*。
- 表单编辑器可以在表单中创建、编辑和删除字段。要创建字段，在“选用板”视图上单击相应的字段类型，然后在编辑器中绘制表示字段大小和位置的矩形。只能在现有表单中添加字段。要编辑字段，单击该字段以选择它，然后使用“属性”视图来编辑它的属性。还可以拖动字段来移动它，或者使用出现在所选表单边框上的调整大小控制柄来调整它的大小。右键单击某个字段以打开其弹出菜单时，将有许多相同选项可用。请参阅 *创建常量字段或创建变量字段*。
- 过滤器可以阻止在表单编辑器中显示表单，允许您在运行时模拟表单组的外观。要切换过滤器、创建过滤器或编辑过滤器，使用编辑器顶部的过滤器按钮。请参阅 *EGL 表单编辑器中的表单过滤器或创建过滤器*。
- 可以通过使用编辑器顶部的显示选项和通过在“首选项”窗口中设置编辑器的首选项来定制表单编辑器的外观。例如，这些选项可以在表单组上显示网格、提高或降低缩放级别以及显示或隐藏字段中的样本值。请参阅 *EGL 表单编辑器的显示选项或设置 EGL 表单编辑器的首选项*。

相关概念

第 150 页的『EGL 表单编辑器概述』

第 141 页的『FormGroup 部件』

第 142 页的『表单部件』

第 160 页的『EGL 表单编辑器的显示选项』

第 161 页的『EGL 表单编辑器中的表单过滤器』

第 156 页的『EGL 表单编辑器中的表单模板』

相关任务

第 152 页的『创建过滤器』

第 157 页的『创建弹出表单』

- 第 158 页的『创建弹出菜单』
- 第 158 页的『在文本或打印表单中显示记录』
- 第 160 页的『设置 EGL 表单编辑器的首选项』
- 第 156 页的『设置 EGL 表单编辑器选用板条目的首选项』
- 『在 EGL 表单编辑器中创建表单』
- 第 153 页的『创建常量字段』
- 第 154 页的『在打印或文本表单中创建变量字段』

相关参考

- 第 464 页的『EGL 源格式的 FormGroup 部件』
- 第 467 页的『EGL 源格式的表单部件』

创建过滤器

要在 EGL 表单编辑器中创建新的过滤器，遵循下列步骤：

1. 在表单编辑器中打开表单组。
2. 在表单编辑器中，单击**过滤器**按钮。“过滤器”窗口将打开。
3. 在“过滤器”视图中，单击**新建**按钮。“新建过滤器”对话框将打开。
4. 在“新建过滤器”对话框中，输入过滤器的名称并单击**确定**。
5. 通过执行下列一个或多个步骤，选择过滤器处于活动状态时显示的表单：
 - 清除想要被过滤器隐藏的表单旁边的复选框。
 - 选择想要被过滤器显示的表单旁边的复选框。
 - 单击**全部选中**按钮以显示每个表单。
 - 单击**全部不选**按钮以隐藏每个表单。
6. 单击**确定**。

新的过滤器现在是活动的。可使用**过滤器**按钮旁边的列表来切换过滤器。

相关概念

- 第 150 页的『EGL 表单编辑器概述』
- 第 151 页的『使用 EGL 表单编辑器编辑表单组』
- 第 160 页的『EGL 表单编辑器的显示选项』
- 第 161 页的『EGL 表单编辑器中的表单过滤器』

相关任务

- 『在 EGL 表单编辑器中创建表单』

在 EGL 表单编辑器中创建表单

要在 EGL 表单编辑器中创建表单，遵循下列步骤：

1. 在表单编辑器中打开表单组。
2. 在“选用板”视图上，单击**文本表单**或**打印表单**。
3. 在编辑器的表单组中，单击并拖动指示表单大小和形状的矩形。“创建表单部件”窗口将打开。
4. 在“创建表单部件”窗口的**输入部件名称**字段中输入表单的名称。此名称将成为 EGL 源代码中该表单部件的名称。
5. 单击**确定**。

- 单击该表单并在“属性”视图中编辑其属性。
- 适当地将字段添加至表单。请参阅 [创建常量字段](#) 和 [创建变量字段](#)。

还可以根据“选用板”视图中的模板创建表单。这些模板使用预定义外观和字段来创建表单。请参阅 [创建弹出表单](#) 或 [创建弹出菜单](#)。

相关概念

- 第 150 页的『EGL 表单编辑器概述』
- 第 151 页的『使用 EGL 表单编辑器编辑表单组』
- 第 141 页的『FormGroup 部件』
- 第 142 页的『表单部件』
- 第 160 页的『EGL 表单编辑器的显示选项』
- 第 161 页的『EGL 表单编辑器中的表单过滤器』
- 第 156 页的『EGL 表单编辑器中的表单模板』

相关任务

- 第 152 页的『创建过滤器』
- 第 157 页的『创建弹出表单』
- 第 158 页的『创建弹出菜单』
- 第 158 页的『在文本或打印表单中显示记录』
- 『创建常量字段』
- 第 154 页的『在打印或文本表单中创建变量字段』

相关参考

- 第 464 页的『EGL 源格式的 FormGroup 部件』
- 第 467 页的『EGL 源格式的表单部件』

创建常量字段

常量字段显示的是在表单中不会更改的文本字符串。与变量字段不同，EGL 代码不能访问常量字段。要将常量字段插入到表单中，遵循下列步骤：

- 在 EGL 表单编辑器中打开表单组。
- 如果表单组没有表单，则向表单组添加表单。请参阅 [创建表单](#)。
- 在“选用板”视图上，单击要添加的常量字段的类型。在缺省情况下，有下列类型的常量字段可用：

表 7. “选用板”视图中的可用常量字段

字段名称	缺省颜色	缺省强度	缺省突出显示	缺省保护
标题	蓝色	粗体	无	跳过
列标题	蓝色	粗体	无	跳过
标签	青色	常规	无	跳过
指示信息	青色	常规	无	跳过
帮助	白色	常规	无	跳过

在基于文本的界面中，这些字段是常用的常量文本字段的样本。可在将它们放在表单上之后定制各个字段。还可以定制“选用板”视图中可用的字段的缺省颜色、强度和突出显示。请参阅 [设置 EGL 表单编辑器选用板条目的首选项](#)。

4. 在编辑器的表单中，单击并按住鼠标以绘制表示字段大小和位置的矩形。鼠标光标旁边的预览框将显示字段的大小及其相对于表单的位置。

注：只能在现有表单中添加字段。

5. 设置好字段的正确大小后，松开鼠标。这就创建了新的字段。
6. 输入想要显示在字段中的文本。
7. 在“属性”视图中，设置新字段的属性。

相关概念

第 150 页的『EGL 表单编辑器概述』

第 151 页的『使用 EGL 表单编辑器编辑表单组』

第 142 页的『表单部件』

相关任务

第 156 页的『设置 EGL 表单编辑器选用板条目的首选项』

『在打印或文本表单中创建变量字段』

相关参考

第 467 页的『EGL 源格式的表单部件』

在打印或文本表单中创建变量字段

变量字段可充当表单中的输入或输出文本。每个变量字段都是基于 EGL 基本部件或 DataItem 部件的。与常量字段不同，EGL 代码可以访问变量字段。要将变量字段插入到表单中，遵循下列步骤：

1. 在 EGL 表单编辑器中打开表单组。
2. 如果表单组没有表单，则向表单组添加表单。请参阅创建表单。
3. 在“选用板”视图上，单击要添加的变量字段的类型。在缺省情况下，有下列类型的变量字段可用：

表 8. “选用板”视图中的可用变量字段

字段名称	缺省颜色	缺省强度	缺省突出显示	缺省保护
输入	绿色	常规	加下划线	否
输出	绿色	常规	无	跳过
消息	红色	粗体	无	跳过
密码	绿色	不可视	无	否

在基于文本的界面中，这些字段是常用的变量文本字段的样本。可在将它们放在表单上之后定制各个字段。还可以定制“选用板”视图中可用的字段的缺省颜色、强度和突出显示。请参阅设置 EGL 表单编辑器选用板条目的首选项。

4. 在编辑器的表单中，单击并按住鼠标以绘制表示字段大小和位置的矩形。鼠标光标旁边的预览框将显示字段的大小及其相对于表单的位置。

注：只能在现有表单中添加字段。

5. 设置好字段的正确大小后，松开鼠标。“新建 EGL 字段”窗口将打开。
6. 在“新建 EGL 字段”窗口的名称字段中输入新字段的名称。
7. 要选择字段类型，执行下列其中一个操作：

- 要使用基本类型，从**类型**列表中单击基本类型。
- 要使用 DataItem 部件，遵循下列步骤：
 - a. 从**类型**列表中单击 **dataitem**。“选择 DataItem 部件”窗口将打开。
 - b. 在“选择 DataItem 部件”窗口中，从列表中单击 DataItem 部件或输入 DataItem 部件的名称。
 - c. 单击**确定**。
- 8. 如果必要，在**维**字段中输入值来设置新变量字段的维。
- 9. 如果想要使字段成为数组，则选择**数组**复选框。
- 10. 如果选择了**数组**复选框，则单击**下一步**并继续执行这些步骤。否则，单击**完成**并停止执行这些步骤。将创建新字段，并且您不必执行余下步骤，这是因为仅当您创建数组时余下步骤才适用。
- 11. 在“新建 EGL 字段”窗口的“数组属性”页面中，在**数组大小**字段中输入数组的大小。
- 12. 从选择下标方向按钮中选择**纵向**或**横向**方向。
- 13. 在**布局的字段纵向排列**和**字段横向排列**字段中输入垂直方向字段和水平方向字段的数目。
- 14. 在**空格的各行之间的空行**和**各列之间的空格**字段中输入数组的各行之间以及各列之间的空格数目。
- 15. 单击**完成**。这就在表单组中创建了新的字段。

创建新字段之后，单击该字段以选择该字段并在“属性”视图中设置它的属性。

因为变量字段没有缺省值，所以这些字段在没有突出显示的情况下可能是不可视的。要使用相应的样本文本标记每个变量字段，在编辑器的顶部单击**切换样本值**按钮。

创建变量字段之后，可在编辑器中双击它以打开“编辑类型属性”窗口。从此窗口中，可以用下列方法编辑该字段：

- 通过在**字段名称**字段中输入新名称来更改字段的名称。
- 从**类型**列表中选择新的字段类型。
- 通过在**精度**字段中输入新的数字来更改字段精度。

在“编辑类型属性”窗口中编辑完字段的属性后，单击**确定**。

相关概念

第 150 页的『EGL 表单编辑器概述』

第 151 页的『使用 EGL 表单编辑器编辑表单组』

第 142 页的『表单部件』

相关任务

第 156 页的『设置 EGL 表单编辑器选用板条目的首选项』

第 153 页的『创建常量字段』

相关参考

第 467 页的『EGL 源格式的表单部件』

设置 EGL 表单编辑器选用板条目的首选项

EGL 表单编辑器选用板的首选项控制选用板中的常量和变量字段类型的缺省颜色、强度和突出显示。要更改这些首选项，遵循下列步骤：

1. 从菜单栏中单击 **窗口 > 首选项**。““首选项”窗口将打开。
2. 在“首选项”窗口的左边窗格中，展开 **EGL > EGL 表单编辑器**，然后单击 **EGL 选用板条目**。
3. 在“首选项”窗口的右边窗格中，对于**选用板条目**列表中的每种类型的常量和变量字段，选择下列选项：
 - 从**颜色**列表中，单击用于该类型的字段的缺省颜色。
 - 从**强度**列表中，单击用于该类型的字段的缺省强度。
 - 从**突出显示**单选按钮中，单击用于该类型的字段的缺省突出显示样式。
 - 从**保护**单选按钮中选择缺省情况下是否保护该字段并且不允许用户进行更新。有关保护字段的更多信息，请参阅 *ConsoleField 属性和字段*。

注：通过单击**复原缺省值**，可以将所有选用板条目复原为其缺省设置。

4. 完成设置选用板条目的首选项后，单击**确定**。

相关概念

第 150 页的『EGL 表单编辑器概述』

第 151 页的『使用 EGL 表单编辑器编辑表单组』

相关任务

第 160 页的『设置 EGL 表单编辑器的首选项』

第 153 页的『创建常量字段』

第 154 页的『在打印或文本表单中创建变量字段』

相关参考第 401 页的『ConsoleField 属性和字段』

EGL 表单编辑器中的表单模板

EGL 表单编辑器使用模板来创建常用的表单和字段类型。这些模板列示在“选用板”视图的**模板抽屉**中。

表单编辑器可以从模板中创建表单。这些表单具有现成的边框、小节和字段。要从模板创建表单，请参阅**创建弹出表单**或**创建弹出菜单**。

表单编辑器可以通过将 EGL 记录用作模板来创建一组字段。要从 EGL 记录创建表示数据的字段，请参阅**在表单中显示记录**。

相关概念

第 150 页的『EGL 表单编辑器概述』

第 151 页的『使用 EGL 表单编辑器编辑表单组』

相关任务

第 157 页的『创建弹出表单』

第 158 页的『创建弹出菜单』

第 158 页的『在文本或打印表单中显示记录』

第 160 页的『设置 EGL 表单编辑器的首选项』

第 156 页的『设置 EGL 表单编辑器选用板条目的首选项』
第 152 页的『在 EGL 表单编辑器中创建表单』

创建弹出表单

弹出表单是可添加至表单组的一种特殊表单。从根本上说，弹出表单与普通文本表单相同，但弹出表单是使用现成的功能部件（如边框和部分）创建的。要在 EGL 表单编辑器中创建弹出表单，遵循下列步骤：

1. 在表单编辑器中打开表单组。
 2. 在“选用板”视图上，单击**弹出表单**。
 3. 在编辑器的表单组中，单击并拖动指示弹出表单大小和形状的矩形。“创建表单部件”窗口将打开。
 4. 在“创建表单部件”窗口的**输入部件名称**字段中输入表单的名称。此名称将成为 EGL 源代码中该表单部件的名称。
 5. 单击**确定**。“新建弹出表单模板”窗口将打开。
 6. 在“新建弹出表单模板”窗口中，在**垂直字符**和**水平字符**字段中输入用于表单边框的字符。
 7. 在**颜色**列表中单击边框的颜色。
 8. 在**强度**列表中单击边框的强度。
 9. 单击**突出显示**单选按钮中的突出显示值。
 10. 对想要添加至表单的每个部分重复下列步骤。必须至少将一个部分添加至表单。
 - a. 在**弹出部分**中，单击**添加**按钮。“创建弹出表单部分”窗口将打开。
 - b. 在“创建弹出表单部分”窗口中，在**部分名称**字段中输入该部分的名称。
 - c. 在**行数字**字段中输入该部分中的行数。输入的数字不要超出余下有效行数，行数显示在“新建弹出表单模板”窗口的底部。
 - d. 单击**确定**。
 - e. 使用**向上**和**向下**按钮来设置字段顺序。
- 注：**弹出字段的各个部分中的总行数不能超出弹出字段中的总行数。在添加部分时，注意**剩余有效行数字**段，并且记住各个部分之间的分隔符还要占用一行以供每个新字段使用。
11. 完成将部分添加至弹出字段之后，单击**完成**。这就在编辑器中创建了新的弹出表单。
 12. 适当地将字段添加至表单。请参阅**创建常量字段**和**创建变量字段**。

相关概念

第 150 页的『EGL 表单编辑器概述』
第 151 页的『使用 EGL 表单编辑器编辑表单组』
第 156 页的『EGL 表单编辑器中的表单模板』

相关任务

第 160 页的『设置 EGL 表单编辑器的首选项』
第 152 页的『在 EGL 表单编辑器中创建表单』
第 153 页的『创建常量字段』
第 154 页的『在打印或文本表单中创建变量字段』
第 158 页的『创建弹出菜单』

创建弹出菜单

弹出菜单是可添加至表单组的一种特殊表单。从根本上说，弹出菜单与普通文本表单相同，但弹出菜单是使用现成的功能部件（如标题、帮助文本和指定数目的菜单选项）创建的。要在 EGL 表单编辑器中创建弹出菜单，遵循下列步骤：

1. 在表单编辑器中打开表单组。
2. 在“选用板”视图上，单击**弹出菜单**。
3. 在编辑器的表单组中，单击并拖动指示弹出菜单大小和形状的矩形。“创建表单部件”窗口将打开。
4. 在“创建表单部件”窗口的**输入部件名称**字段中输入表单的名称。此名称将成为 EGL 源代码中该表单部件的名称。
5. 单击**确定**。“新建弹出菜单模板”窗口将打开。
6. 在“新建弹出菜单模板”中，在**宽度**和**高度**字段中输入弹出菜单的大小。在缺省情况下，将使用您在编辑器中创建的表单大小填充这些字段。
7. 在**菜单标题**字段中，输入菜单的标题。
8. 在**菜单选项数字**字段中，输入弹出菜单将会包含的菜单选项的数目。
9. 在**菜单帮助文本**字段中，输入菜单的所有附加帮助文本。
10. 单击**完成**。这就在编辑器中创建了新的弹出菜单。
11. 适当地将字段添加至新的弹出菜单并编辑现有字段。请参阅**创建常量字段和创建变量字段**。

相关概念

第 150 页的『EGL 表单编辑器概述』

第 151 页的『使用 EGL 表单编辑器编辑表单组』

第 156 页的『EGL 表单编辑器中的表单模板』

相关任务

第 160 页的『设置 EGL 表单编辑器的首选项』

第 152 页的『在 EGL 表单编辑器中创建表单』

第 153 页的『创建常量字段』

第 154 页的『在打印或文本表单中创建变量字段』

第 157 页的『创建弹出表单』

在文本或打印表单中显示记录

“选用板”视图的**模板抽屉**中的**记录模板**创建一组表单字段，这些字段相当于 EGL 记录部件中的那些字段。要创建表单字段，遵循下列步骤：

1. 在 EGL 表单编辑器中打开表单组。
2. 创建表单。请参阅在 *EGL 表单编辑器中创建表单*。
3. 在“选用板”视图上，单击**记录**。
4. 在编辑器的表单中，单击并按住鼠标以绘制表示字段大小和位置的矩形。鼠标光标旁边的预览框将显示记录字段的大小以及它们相对于字段的位置。

注：只能在现有表单中添加记录。

5. 设置好记录的正确大小后，松开鼠标。“放置 EGL 记录”窗口将打开。
6. 在“放置 EGL 记录”中，单击**浏览**。“选择记录部件”对话框将打开。

7. 在“选择记录部件”对话框中，单击想要使用的记录部件的名称或输入记录部件的名称。
8. 单击**确定**。现在就会使用该记录中的一系列字段填充“创建记录”窗口。
9. 使用下列其中一个或多个方法，选择并组织想要显示为表单中的字段的记录部件字段：
 - 要除去字段，单击它的名称，然后单击**除去**。
 - 要添加字段，遵循下列步骤：
 - a. 单击**添加按钮**。“编辑表条目”窗口将打开。
 - b. 在“编辑表条目”窗口的**字段名称框**中输入字段的名称。
 - c. 在**类型**列表中选择字段的类型。
 - d. 对于您选择的类型，如果必要，请在**精度**字段中输入字段的精度。
 - e. 在**字段宽度**字段中输入字段的宽度。
 - f. 如果想要字段成为输入字段，则选择**使此字段成为输入字段**复选框。否则清除该复选框。
 - g. 单击**确定**。
 - 要编辑字段，遵循下列步骤：
 - a. 单击字段的名称。
 - b. 单击**编辑按钮**。“编辑表条目”窗口将打开。
 - c. 在“编辑表条目”窗口的**字段名称框**中输入字段的名称。
 - d. 在**类型**列表中选择字段的类型。
 - e. 对于您选择的类型，如果必要，请在**精度**字段中输入字段的精度。
 - f. 在**字段宽度**字段中输入字段的宽度。
 - g. 如果想要字段成为输入字段，则选择**使此字段成为输入字段**复选框。否则清除该复选框。
 - h. 单击**确定**。
 - 要在列表中上下移动字段，使用**向上**和**向下**按钮。
10. 使用**方向**单选按钮来为字段选择垂直方向或水平方向。
11. 在**行数字段**中输入想要这一组字段包含的行数。
12. 如果想要这一组字段包含标题行，则选择**创建标题行**复选框。
13. 单击**完成**。

相关概念

- 第 150 页的『EGL 表单编辑器概述』
- 第 151 页的『使用 EGL 表单编辑器编辑表单组』
- 第 156 页的『EGL 表单编辑器中的表单模板』

相关任务

- 第 152 页的『在 EGL 表单编辑器中创建表单』
- 第 153 页的『创建常量字段』
- 第 154 页的『在打印或文本表单中创建变量字段』

EGL 表单编辑器的显示选项

EGL 表单编辑器具有显示选项，允许您控制表单组在设计时出现在编辑器中的方式。这些选项在运行时不会更改表单组的外观。编辑器顶部左端到右端是控制显示选项的按钮：

切换网格线

此选项在表单组上显示网格以帮助调整表单的大小并排列它们。要更改网格的颜色，请参阅设置 *EGL* 表单编辑器的首选项。

切换样本值

此选项将样本值插入到本来不可视的变量字段中。

切换黑色和白色方式

此选项将编辑器的背景从黑色切换为白色。

缩放级别

设置编辑器的放大级别。

编辑器顶部还有其它按钮，用来控制表单组和编辑器的过滤器的大小。请参阅使用 *EGL* 表单编辑器编辑表单组或 *EGL* 表单编辑器中的表单过滤器。

相关概念

第 150 页的『*EGL* 表单编辑器概述』

第 161 页的『*EGL* 表单编辑器中的表单过滤器』

相关任务

第 151 页的『使用 *EGL* 表单编辑器编辑表单组』

第 152 页的『创建过滤器』

『设置 *EGL* 表单编辑器的首选项』

设置 EGL 表单编辑器的首选项

EGL 表单编辑器的首选项可以更改表单编辑器的外观，如背景色和网格颜色。要更改表单编辑器的首选项，遵循下列步骤：

1. 从菜单栏中单击 **窗口 > 首选项**。““首选项”窗口将打开。
2. 在“首选项”窗口的左边窗格中，展开 **EGL**，然后单击 **EGL 表单编辑器**。
3. 在“首选项”窗口的右边窗格中，选择表单编辑器的首选项：
 - 在 **背景色** 字段中，选择表单编辑器的背景色。
 - 在 **网格颜色** 字段中，选择表单编辑器的网格颜色。
 - 如果想要在字段周围显示边框，则选择 **突出显示字段** 复选框并选择颜色。
 - 如果想要在表单编辑器的顶部和左边显示标尺，则选择 **显示标尺** 复选框。
 - 在 **字体** 列表中，单击用于字段的字体并从接近的列表中单击某个大小。

注： 选择等宽字体以确保字段在表单编辑器中以正确的大小显示。等宽字体就是所有字符的宽度相同的字体，如 *Courier New*。

- 如果想要闪烁字段在编辑器中以斜体类型显示，则选择 **以可视方式说明闪烁字段** 复选框。此选项不会在运行时更改字段的外观，只会在设计时更改字段的外观。

注： 通过单击 **复原缺省值**，可将 *EGL* 表单编辑器首选项窗口复原为其缺省设置。

4. 完成设置 *EGL* 表单编辑器的首选项后，单击 **确定**。

相关概念

第 150 页的『EGL 表单编辑器概述』

第 151 页的『使用 EGL 表单编辑器编辑表单组』

『EGL 表单编辑器中的表单过滤器』

相关任务

第 156 页的『设置 EGL 表单编辑器选用板条目的首选项』

EGL 表单编辑器中的表单过滤器

过滤器限制显示在 EGL 表单编辑器中的表单。可定义任意数目的过滤器，但一次只能有一个过滤器是活动的。过滤器只影响设计时的表单组表示；对 EGL 代码没有任何影响。请参阅[创建过滤器](#)。

可使用编辑器顶部的[过滤器](#)按钮旁边的列表切换活动的过滤器。要创建、编辑或删除过滤器，单击[过滤器](#)按钮。

从“过滤器”窗口中，可以使用下列功能来管理过滤器：

- 从列表中选择过滤器。
- 通过单击[新建](#)来添加新的过滤器。
- 通过从列表中选择过滤器然后单击[除去](#)来删除过滤器。
- 选择过滤器处于活动状态时显示的表单。

相关概念

第 150 页的『EGL 表单编辑器概述』

第 151 页的『使用 EGL 表单编辑器编辑表单组』

相关任务

第 152 页的『创建过滤器』

第 152 页的『在 EGL 表单编辑器中创建表单』

创建控制台用户界面

控制台用户界面

控制台用户界面（ConsoleUI）是在 Windows 或 UNIX 屏幕上以基于文本的格式显示数据的一种技术。此技术只能在 EGL 生成的 Java 程序中使用，而不能在 PageHandler 中使用。

使用 ConsoleUI 创建的界面可在本地或通过远程终端会话显示在 Windows 2000/NT/XP 或 UNIX X-windows 中。

ConsoleUI 与文本用户界面（TextUI）不同，两者不能在同一程序中运行：

- 如果 TextUI 生效，则界面样式类似在与 3270 终端交互的大型机程序中使用的界面样式。该程序显示文本表单，但在用户从一个字段移至下一个字段时不会处理用户输入。当用户提交表单（在大多数情况下是通过按 **Enter** 键进行的），表单中的所有数据将返回至程序，只有此时程序才会验证数据；如果验证成功，程序将运行接下来的编码语句。
- 如果 ConsoleUI 生效，则界面样式类似在与基于字符的终端交互的基于 UNIX 的程序中使用的界面样式。程序显示控制台表单，并且在用户按 **Tab** 键以将屏幕上的光标移至下一个字段时立即响应用户事件。验证是按逐个字段进行的，您可以将光标限制在当前字段直到用户在其中输入了有效的数据。

使用 consoleUI 时，通常会编写如下程序：

1. 声明基于一直可用的 ConsoleUI 部件的一组变量；不能定义特定于 ConsoleUI 的部件。
2. 通过在调用相应的 EGL 函数时将 consoleUI 变量作为自变量加入来打开可视实体，如表单。或者，可以通过调用 EGL 函数（如 **displayFormByName**，它接受在运行时已知的名称）来打开可视实体。
3. 在 EGL **openUI** 语句中引用可视实体，这允许通过输入特定事件（如用户击键）以获取特定逻辑来进行用户交互。

consoleUI 应用程序的用户可以按键来与屏幕上显示的内容进行交互，但单击鼠标不起作用。

ConsoleUI 可以接受用户输入到字段中的内容，但仅当指定绑定（表示输入字段与基本类型的变量之间的一致）时才会如此。EGL 运行时执行下列操作：

- 将变量值用作已显示字段的初始内容；并且
- 在用户离开该字段时将用户输入移至该变量。

ConsoleUI 还允许您以行式与用户交互，行式是代码一次只读写一行的方式。行式的含义包括：

- 在 Eclipse 工作台中，用户与“控制台”视图交互
- 在使用命令提示符调用的程序中，用户与命令窗口交互

- 在 UNIX 中的 Curses 下运行的程序中，用户与显示用户界面的窗口交互；一般情况下，基于窗口的交互是被禁止的

ConsoleUI 相当于 Informix 4GL 产品中的用户界面技术。

相关任务

『使用 consoleUI 创建界面』

相关参考

第 165 页的『ConsoleUI 部件和相关变量』

第 169 页的『UNIX 的 ConsoleUI 屏幕选项』

第 691 页的『EGL 库 ConsoleLib』

第 565 页的『openUI』

第 168 页的『ConsoleUI 中 new 的用法』

使用 consoleUI 创建界面

控制台用户界面 (ConsoleUI) 是在 Windows 或 UNIX 屏幕上以基于文本的格式显示数据的一种技术。

使用 consoleUI 创建界面的步骤如下所示:

1. 创建 EGL 源文件
2. 编写包括 *ConsoleUI* 部件和相关变量中描述的语言元素的程序
3. 从 EGL 源文件生成 Java 代码
4. 将生成的 Java 文件作为应用程序运行

其中每个任务将在下面详细描述。

创建 EGL 源文件

1. 在工作台中，从 EGL 透视图中选择文件 > 新建 > EGL 源文件。或者从任意透视图中选择文件 > 新建 > 其它 > EGL 源文件。
2. 在向导屏幕中输入以下信息：
 - **源文件夹**: 将包含 EGL 源文件的目录位置。
 - **包**: 将包含 EGL 源文件的包位置。此字段是可选的。
 - **EGL 源文件名称**: 控制台用户界面源文件的文件名，如 **myConsoleUI**。
3. 选择**完成**以创建该文件。扩展名 (**.egl**) 将自动追加至文件名的末尾。EGL 文件便出现在“项目资源管理器”视图中，并在缺省 EGL 编辑器中自动打开。

编写 ConsoleUI 程序

要填充源文件并创建 ConsoleUI，需要使用 ConsoleUI 语言元素，它们是在 **egl.ui.console** 概述帮助主题中引入的，并且是在各个 ConsoleUI 库、**OpenUI** 语句、记录类型和枚举帮助主题中作了完整定义的。

ConsoleUI 应用程序必须至少包括下列元素:

1. PROGRAM...END
2. Function main()
3. OpenUI 语句

注：尽管 **OpenUI** 语句对于 ConsoleUI 非常基础，但您可以在没有 **OpenUI** 语句的情况下成功编写 ConsoleUI 程序。

从 EGL 源代码生成 Java 代码

要生成 Java 文件：

1. 在 EGL 编辑器中，右键单击 ConsoleUI 文件。将显示上下文菜单。
2. 选择生成。

注：ConsoleUI **.egl** 源文件不能生成为 COBOL。

将生成的 Java 文件作为应用程序运行

要运行生成的 Java 文件：

1. 从“项目资源管理器”中，右键单击生成的 Java (**.java**) 文件。将显示上下文菜单。
2. 选择运行 > 运行方式 > **Java 应用程序**。
3. 或者，在 Java 文件在编辑器中打开的情况下，从主菜单选择运行 > 运行方式 > **Java 应用程序**。
4. ConsoleUI 将显示至窗口。

ConsoleUI 应用程序可在基于 `curse` 的终端会话或基于 `Swing` 的图形窗口中显示。UNIX 用户可以选择更灵活的显示方式，在 *UNIX 的 ConsoleUI 屏幕选项帮助主题* 中对此作了描述。

注：IBM 不支持在同一程序中同时使用 ConsoleUI 和 TextUI。

相关概念

第 163 页的『控制台用户界面』

相关参考

第 691 页的『EGL 库 ConsoleLib』

『ConsoleUI 部件和相关变量』

第 169 页的『UNIX 的 ConsoleUI 屏幕选项』

第 565 页的『openUI』

ConsoleUI 部件和相关变量

使用 consoleUI 时，创建下列种类的变量，这些变量基于相关的 consoleUI 部件：

- Window
- Prompt
- ConsoleField
- ConsoleForm
- Menu
- MenuItem

库 **ConsoleLib** 还包括类型为 `PresentationAttributes` 的系统变量。系统变量控制已显示输出的可视外观；并且要更改显示的外观，可通过设置 `PresentationAttributes` 字段

color、**highlight** 和 **intensity** 来更改这些变量。有关这些字段的详细信息，请参阅 *EGL consoleUI* 中的 *PresentationAttributes* 字段。

Window (窗口)

窗口是一个矩形区域，可在其中放置表示变量的其它可视实体。

在显示一个窗口而其它窗口都未在工作时，新窗口将在屏幕窗口中，这是一个矩形，具有操作系统中任何窗口的基本特征。在使用 *Curses* 库时，对于 UNIX 会有所不同；在这种情况下，*consoleUI* 窗口的显示将现有终端窗口置于开窗方式。

显示的任何其它窗口出现在屏幕窗口的内容部分，通常在已经打开的窗口的顶部。这些窗口还可以并排显示。

声明窗口时，可以设置各种属性。例如，**position** 是相对于屏幕的左上角的位置；而 **size** 是窗口的高度和宽度（以字符数计）。

以下是窗口声明的一个示例：

```
myWindow WINDOW
{name="myWindow", position = [2,2],
 size = [18,75], color = red, hasborder=yes};
```

通过使用名称以 *ConsoleLib.openWindow* 开头的 EGL 函数显示窗口。如果在显示其它数据时未显示窗口，EGL 将提供一个窗口。

Prompt (提示)

提示是引出用户输入的一行语句。提示的声明如下所示：

```
myPrompt Prompt { message = "Type your ID: "};
```

通过将变量包括在 **openUI** 语句中来显示提示，该语句会将提示与类型为字符串的变量绑定，但仅适用于输入。可将提示配置为接受单个字符或字符串。

ConsoleField

consoleField 是屏幕上的字段，它是在控制台表单的上下文中声明的（如后文所述）。下一个示例声明 *consoleField*，它的内容在运行时可能会有变化：

```
myField ConsoleField (
  name="myFieldName",
  position=[1,31],
  fieldLen=20,
  binding = "myVariable" );
```

要指定常量文本，使用星号 (*) 来代替变量名，如以下示例中所示：

```
* ConsoleField
{ position=[2,5], value="Title: "};
```

强烈建议在声明指定 *consoleField* 时使用同一名称来表示 *consoleField* 和 *consoleField* 中的名称属性的值。但是，不同的名称对这两种用法也有效。在生成时解析对 *consoleField* 的访问时，应引用 *consoleField* 名称（如 *myField*）。在运行时解析访问时应引用名称属性值（如 *myFieldName*），就像使用 *consoleField* 在 **openUI** 语句中定义事件时那样。

ConsoleForm

consoleForm 主要由一组 consoleField 组成。要使 consoleForm 处于活动状态，调用系统函数 **ConsoleLib.displayForm**。例如，要显示只读 consoleForm，执行下列操作：

1. 调用 **ConsoleLib.displayForm**
2. 调用系统函数 **ConsoleLib.getKey** 来等待用户击键

要允许用户写入 consoleField，则执行下列操作：

1. 调用 **ConsoleLib.displayForm**
2. 发出引用显示的 consoleForm 或 consoleForm 中的特定 consoleField 的 **openUI** 语句。

consoleForm 是子类型为 ConsoleForm 的记录，不仅可以包括 consoleField，还可以包括在任何 EGL 记录中有效的任何字段。

要允许用户与屏幕上的 consoleField 表交互，执行下列操作：

1. 在 consoleForm 中，声明依次引用 consoleField 数组的 arrayDictionary，这些 consoleField 数组也是在 consoleForm 中声明的
2. 在 **openUI** 语句中使用该 arrayDictionary

如果只允许用户与 consoleForm 中的一部分 consoleField 交互，可以在 **openUI** 语句中以显式方式或通过引用字典来列出这些 consoleField。与 arrayDictionary 一样，该字典是在 consoleForm 中声明的，并且引用同样在该 consoleForm 中声明的 consoleField。

EGL 不显示在 consoleForm 中声明的任何基本变量。可使用这种变量来绑定 consoleField，就像可以使用在 consoleForm 外部声明的变量来绑定一样。

一般来说，可以下列两种方法中的任何一种来创建 consoleForm 绑定：

- 通过在声明 consoleForm 时设置缺省绑定。
- 通过在编写 **openUI** 语句时设置绑定。

在 **openUI** 语句中指定的任何绑定将完全覆盖缺省绑定；不会保留任何 consoleForm 声明绑定。

如果使用 **openUI** 语句来绑定变量，一个选择是使用语句属性 **isConstruct**，这将执行下列操作：

- 将用户输入的格式定义为适用于 SQL WHERE 子句的字符串
- 将该字符串放在单个变量中，这样就可以很方便地编写 SQL SELECT 语句，该语句将从关系数据库检索用户请求的数据，就像在编写 EGL **prepare** 语句时一样

有关属性 **isConstruct** 的详细信息，请参阅 *OpenUI 语句*。

跳进顺序是用户从一个 consoleField 跳进另一 consoleField 的顺序。在缺省情况下，跳进顺序就是 consoleForm 声明中的 consoleField 顺序。如果在 **openUI** 语句中输入 consoleField 列表，则跳进顺序就是该语句中的 consoleField 顺序；同样，如果在 **openUI** 语句中输入字典或 arrayDictionary，则跳进顺序就是该字典或 arrayDictionary 的声明中的 consoleField 顺序。

在缺省情况下，用户可通过按 **Esc** 键来退出与 consoleForm 有关的 **openUI** 语句。

Menu (菜单)

菜单是水平显示的一组标签。一个标签表示整个菜单，一个标签表示菜单中的每个 `menuItem`。要确保用户选择特定 `menuItem` 时发生响应，在 `openUI` 语句中引用整个菜单并在该语句的 `OnEvent` 子句中引用该 `menuItem`。

MenuItem

`menuItem` 显示一个标签，使用方法在上一节中作了描述。

相关概念

第 79 页的『ArrayDictionary』
第 163 页的『控制台用户界面』
第 75 页的『字典』

相关参考

第 401 页的『ConsoleField 属性和字段』
第 413 页的『EGL consoleUI 中的 ConsoleForm 属性』
第 691 页的『EGL 库 ConsoleLib』
第 169 页的『UNIX 的 ConsoleUI 屏幕选项』
第 414 页的『EGL consoleUI 中的 Menu 字段』
第 415 页的『EGL consoleUI 中的 MenuItem 字段』
第 565 页的『openUI』
第 417 页的『EGL consoleUI 中的 PresentationAttributes 字段』
第 419 页的『EGL consoleUI 中的 Prompt 字段』
第 420 页的『EGL consoleUI 中的 Window 字段』

相关任务

第 164 页的『使用 consoleUI 创建界面』

ConsoleUI 中 new 的用法

在创建使用 `consoleUI` 的 EGL 程序时，类型为 `Menu`、`MenuItem`、`Prompt` 和 `Window` 的每个变量都是引用变量，它包含的内存地址引用了存储在该变量外部的值。

可以像声明任何其他变量那样声明引用变量，如以下示例中所示：

```
myPrompt Prompt { message = "Type your ID: "};
```

或者，可以声明引用变量并使用保留字 `new` 来进行初始化，如以下示例中所示：

```
myPrompt Prompt = new Prompt { message = "Type your ID: "};
```

在声明变量时，两种格式之间的差别没有什么实际影响；但在编写 `openUI` 语句时，在编码时使用 `new` 这个字会比较方便，如 `openUI` 中所示。

`new` 的常规语法如下所示：

```
new partName
```

partName

下列其中一个字，它指的是特定种类的部件：

- `Menu`
- `MenuItem`

- Prompt
- Window

有关引用变量的其它含义的详细信息，请参阅 *EGL* 中的引用兼容性。

相关概念

第 163 页的『控制台用户界面』

相关参考

第 165 页的『ConsoleUI 部件和相关变量』

第 565 页的『openUI』

第 675 页的『EGL 中的引用兼容性』

相关任务

第 164 页的『使用 consoleUI 创建界面』

UNIX 的 ConsoleUI 屏幕选项

受支持 UNIX 平台上的 EGL 用户能够使用图形显示方式或 UNIX *curse* 方式来运行 ConsoleUI 应用程序。

图形显示方式

要以图形显示方式运行 ConsoleUI 应用程序，必须确保 EGL *curse* 库不在正在运行的 shell 的库路径环境变量中。这是缺省方式。

UNIX *curse* 方式

要以 UNIX *curse* 方式运行 ConsoleUI 应用程序，正在运行的 shell 的库路径环境变量中必须具有相应的特定于平台的 EGL *curse* 库。EGL *curse* 库必须从 EGL Support Web 站点下载。

要下载 EGL *curse* 库:

1. 找到相应的 EGL Support Web 站点。

- Rational Application Developer 的 URL 为:

<http://www3.software.ibm.com/ibmdl/pub/software/rationalsdp/rad/60/redist>

- Rational Web Developer 的 URL 为:

<http://www3.software.ibm.com/ibmdl/pub/software/rationalsdp/rwd/60/redist>

2. 将 **EGLRuntimesV60IFix001.zip** 文件下载至您喜欢的目录。

3. 解压缩 **EGLRuntimesV60IFix001.zip** 以标识下列文件:

- AIX®: **EGLRuntimes/Aix/bin/libCursesCanvas6.so**
- Linux: **EGLRuntimes/Linux/bin/libCursesCanvas6.so**

4. 在库路径环境变量中插入相应的 EGL *curse* 库。

AIX: 使用以下 bourne-shell 来设置 '**LIBPATH**' 库路径环境变量:

```
"LIBPATH=$INSTDIR/aix; export LIBPATH"
```

Linux: 使用以下 bourne-shell 来设置 'LD_LIBRARY_PATH' 库路径环境变量:
"LD_LIBRARY_PATH=\$INSTDIR/aix; export LD_LIBRARY_PATH"

相关概念

第 163 页的『控制台用户界面』

相关参考

第 691 页的『EGL 库 ConsoleLib』

第 165 页的『ConsoleUI 部件和相关变量』

第 565 页的『openUI』

相关任务

第 164 页的『使用 consoleUI 创建界面』

创建 EGL Web 应用程序

Web 支持

EGL 按照下列方式提供对基于 Web 的应用程序的支持:

- 您可以开发 *PageHandler*，它是一个逻辑部件，它的每个函数都被 Web 页面中的特定用户操作调用。此逻辑部件的生成也可以提供可以进行定制的 *JavaServer Faces JSP*。
- 您可以提供若干 Web 页面的公共功能，例如，应用程序中的每个页面都显示一个按钮，用户可以通过单击该按钮以注销。在此例中，用户的单击可以调用一个 EGL 程序，该程序充当公共子例程。
- 最后，当您在 *WebSphere Page Designer* 中工作时，可以定制 *JavaServer Faces JSP*，并可以影响 *PageHandler*，如 *EGL 的 Page Designer 支持* 中所述。

相关概念

第 177 页的『*PageHandler*』

第 309 页的『*WebSphere Application Server 和 EGL*』

相关任务

第 308 页的『在本地机器上启动 Web 应用程序』

相关参考

第 175 页的『*EGL 的 Page Designer 支持*』

创建单个表 EGL Web 应用程序

EGL 数据部件和页面向导

“EGL 数据部件和页面”向导使您能够很方便地创建基于 Web 的实用程序，该实用程序允许您维护关系数据库中的特定表。

该向导创建以下实体:

- 一组 *PageHandler*，稍后您会将它们生成到在 *Java Server Faces* 中运行的一组部件中
- 一个 *SQL* 记录部件以及相关的数据项部件和基于库的函数部件
- 一组 *JSP* 文件，它们提供以下 Web 页面:
 - 选择标准页面，它接受用户的选择标准
 - 列表页面，根据用户的条件显示多行
 - 创建详细信息页面，允许用户显示或插入一行
 - 详细信息页面，允许用户显示、更新或删除一行

用户最先遇到选择标准页面；但如果未能指定该页面所需的信息，则用户最先遇到列表页面，它提供对表中每一行的访问（在此情况下）。

在此向导中工作时，可执行下列任务：

- 通过改变显示的字段或添加链接两个页面的链接来定制之前描述的 Web 页面。
- 指定用于从给定数据库表或视图中创建、读取、更新或删除行的 SQL 记录键字段。
- 定制用于创建、读取或更新行的显式 SQL 语句。（不能定制用于删除行的 SQL 语句）。
- 指定用于从给定数据库或视图中选择一组行的 SQL 记录键字段。
- 定制用于选择一组行的显式 SQL 语句。
- 验证并运行每个 SQL 语句

输出包括下列文件：

- 调用 Web 应用程序的 HTML 文件（index.html）。
- 提供之前描述的 Web 页面的一组 JSP 文件。
- 一个 EGL 源文件，该文件包含 SQL 记录部件中的结构项引用的所有数据项部件。
- 对于每个 SQL 记录部件，向导还会产生两个文件，一个文件用于记录部件本身，另一个用于相关的基于库的函数。如果选择了同一文件中的记录 and 库复选框，可以减少文件数目。

可在向导创建基于 Web 的实用程序后定制它。

相关概念

第 297 页的『Java 程序、PageHandler 和库』

第 209 页的『SQL 支持』

相关任务

『创建单个表 EGL Web 应用程序』

第 234 页的『对 EGL 向导创建、编辑或删除数据库连接』

第 235 页的『在 EGL 向导中定制 SQL 语句』

第 174 页的『在 EGL 数据部件和页面向导中定义 Web 页面』

创建单个表 EGL Web 应用程序

要从单个关系数据库表创建 EGL Web 应用程序，执行下列操作：

1. 选择文件 > 新建 > 其它...。将显示用于选择向导的对话框。
2. 展开 **EGL** 并双击 **EGL 数据部件和页面**。将显示“EGL 数据部件和页面”对话框。
3. 输入 EGL Web 项目名，或者从下拉列表中选择现有项目。将在此项目中生成 EGL 部件。
4. 从下拉列表中选择现有数据库连接或建立新的数据库连接：
 - 要建立新的数据库连接，单击**添加**并遵循帮助主题**数据库连接页面**中的指示信息，可通过按 **F1** 键访问该帮助主题。
 - 有关编辑或删除数据库连接的详细信息，请参阅对 *EGL 向导创建、编辑或删除数据库连接*

与数据库建立连接后，将显示数据库表列表。

5. 如果不想接受数据项的缺省 EGL 文件名，输入新的文件名。

6. 在**选择数据**字段中，单击所需的数据库表的名称。
7. 在**记录名字**段中，指定要创建的 EGL 记录名或接受缺省名称。
8. 如果想要将库部件和 SQL 记录部件包括在同一文件中，则选择该复选框。
9. 要将其它字段设置为缺省值以外的值，单击**下一步**；否则单击**完成**。其余的步骤假定您已经单击**下一步**。
10. 选择在读取、更新和删除各行时要使用的关键字段，然后单击向右箭头。要选择多个关键字段，在单击不同字段名时按住 **Ctrl** 键。要从右边的列表中除去关键字段，使该字段名突出显示并单击向左箭头。
11. 选择在选择一组行时要使用的选择条件字段，然后单击向右箭头。要选择多个字段，在单击不同字段名时按住 **Ctrl** 键。要从右边的列表中除去字段，使该字段名突出显示并单击向左箭头。
12. 要定制隐式 SQL 语句，请参阅在 *EGL 向导中定制 SQL 语句*。此选项对 EGL **delete** 语句不可用。
13. 单击**下一步**。
14. 如果想要对新的 Web 页面应用模板，遵循下列步骤：
 - a. 选择**选择页面模板**复选框。
 - b. 通过单击**样本页面模板**或**用户定义的页面模板**来选择页面模板类型。
 - c. 单击想要使用的页面模板。可选择缩略图或通过单击**浏览**按钮来浏览至模板位置。
15. 单击**下一步**。
16. 要定制 Web 页面，请参阅在 *EGL 数据部件和页面向导中定义 Web 页面*。
17. 单击**下一步**。
18. 将显示“生成 Web 应用程序”屏幕，包括（在底部）将产生的文件和 Web 页面的列表：
 - a. 要更改将接收 EGL 部件的 EGL Web 项目的名称，在 **EGL Web 项目名称** 字段中输入项目名称或从相关下拉列表中选择个项目。
 - b. 要对特定类型的部件（PageHandler、数据或库）指定 EGL 和 Java 包，在相关字段中输入包名或从相关下拉列表中选择一名称。
 - c. 要更改为给定 Web 页面产生的 JSP 和 EGL 文件的名称，单击 **Web 页面** 中的相应条目并输入新名称。每个文件名包括您输入的任何字母或数字，但不包括空格和其它字符。

对 PageHandler 部件、数据部件和库部件输入或选择包。
19. 单击**完成**。

相关概念

第 209 页的『SQL 支持』

相关任务

第 234 页的『对 EGL 向导创建、编辑或删除数据库连接』

第 233 页的『从关系数据库表创建 EGL 数据部件』

第 235 页的『在 EGL 向导中定制 SQL 语句』

第 174 页的『在 EGL 数据部件和页面向导中定义 Web 页面』

在 EGL 数据部件和页面向导中定义 Web 页面

“EGL 数据部件和页面”向导从关系数据库表创建 Web 应用程序。在此向导中工作时，可指定产生的每个 Web 页面类型的各个方面：

- 页面标题
- 样式表
- 要显示的字段，包括它们的顺序和属性
- 与其它页面的链接

在称为定义应用程序的 Web 页面的向导对话框中工作时，可单击选项卡来浏览页面。对每个页面执行下列操作（只要可能）：

1. 在**页面标题**字段中设置页面标题。
2. 从**样式表**字段中的下拉列表中选择样式表。
3. 要查看接受当前页面定义的效果，单击**预览**。
4. 如果希望指定要在一个页面上显示的行数，选择**分页**并对**页大小**指定行数（正整数）。
5. 选择要包括在页面上的字段：
 - a. 要包括字段，选择相关的复选框。要选择每个字段，单击**全部**。
 - b. 要排除字段，清除相关的复选框。要排除每个字段，单击**无**。
 - c. 要更改字段的显示位置，单击该字段，然后使用向上和向下箭头将该字段移至另一位置。
 - d. 要设置某个字段的属性，单击该字段然后双击“属性”窗格中的**值**字段。输入值，或者在某些情况下从下拉列表中进行选择。
6. 选择用户可在页面上执行的操作：
 - a. 要包括某个操作，选择相关的复选框。要选择每个操作，单击**全部选中**。

这些操作包括**创建**、**删除**、**抽取**、**列示**和**读取**：

- **创建**链接至创建详细信息页面，用户可在其中显示或插入一行。该选项显示在创建详细信息页面上只是说明了用户可从该页面创建行。
 - **删除**仅在详细信息页面上可用。此选项删除具有用户指定的键的记录。
 - **抽取**链接至选择条件页面，它接受用户的选择标准。该选项显示在选择条件页面上只是说明了用户可以让该页面返回结果集。
 - **列示**链接至列示页面，它根据用户的条件显示多行。
 - **读取**仅在创建详细信息页面上可用。此选项显示具有用户指定的键的记录。
 - **更新**仅在详细信息页面上可用。此选项将更新用户修改过的记录。
- b. 要排除操作，清除相关的复选框。要排除每个操作，单击**无**。

如果给定操作一直可用，则不能清除该复选框。

- c. 要设置操作的 Web 页面标注，单击操作名，然后双击“属性”窗格中的**值**字段并输入值。

相关概念

第 209 页的『SQL 支持』

第 171 页的『EGL 数据部件和页面向导』

相关任务

第 172 页的『创建单个表 EGL Web 应用程序』

第 233 页的『从关系数据库表创建 EGL 数据部件』

第 234 页的『对 EGL 向导创建、编辑或删除数据库连接』

第 105 页的『设置 EGL 首选项』

第 308 页的『在本地机器上启动 Web 应用程序』

创建 EGL pageHandler 部件

pageHandler 部件通过为 Java Server Faces JSP 提供数据和服务来控制用户与 Web 页面的运行时交互。在创建 JSP 时，将自动在 *EGLSource* 文件夹中名为 *pagehandlers* 的包中创建 pageHandler 部件。pageHandler 部件的名称与对应的 JSP 相同，但该部件具有 .egl 文件扩展名。

可选择创建 pageHandler 部件，并且如果 EGL Web 项目中还没有同名的 JSP 文件，可让系统自动将 JSP 添加至项目。要创建 EGL pageHandler 部件，执行下列操作：

1. 如果 EGL Web 项目中没有包含名为 *pagehandlers* 的包，则必须创建它。Page Designer 要求所有 pageHandler 部件位于名为 *pagehandlers* 的包中。有关创建包的详细信息，请参阅[创建 EGL 包](#)。
2. 在要包含 pageHandler 部件的 *pagehandlers* 包中标识 EGL 文件。在 EGL 编辑器中打开该文件。如果还没有 EGL 文件，则必须创建它。
3. 根据 EGL 语法输入 pageHandler 部件的细节（有关详细信息，请参阅[EGL 源代码格式的 PageHandler 部件](#)）。可使用内容辅助以将 pageHandler 部件语法的概要内容放在文件中。
4. 保存 EGL 文件。

相关概念

第 13 页的『EGL 项目、包和文件』

第 177 页的『PageHandler』

相关任务

第 117 页的『创建 EGL 包』

第 118 页的『创建 EGL 源文件』

第 118 页的『将 EGL 模板与内容辅助配合使用』

第 184 页的『对 PageHandler 代码使用“快速编辑”视图』

相关参考

第 441 页的『EGL 中的内容辅助』

第 612 页的『命名约定』

第 619 页的『EGL 源格式的 PageHandler 部件』

EGL 的 Page Designer 支持

在 EGL Web 项目中创建 JSP 文件时，EGL 会自动创建 PageHandler，并且该 PageHandler 将包括要定制的 EGL 框架代码。然后在 Page Designer 中执行下列任务：

1. 将组件从选用板中拖到 JSP 中
2. 使用“属性”视图设置颜色之类特定于组件的特征并设置绑定，绑定是组件与数据或逻辑之间的关系

可执行特定于 EGL 的工作:

- 创建 EGL 变量并将它们放到现有 PageHandler 中。
- 将 PageHandler 项与 JSP 用户界面组件绑定。
- 将 PageHandler 函数与按钮和超链接控件绑定。这些函数将充当事件处理程序。

在 Page Designer 中使用源代码选项卡时, 可手工将 JSP 文件 (具体地说, JavaServer Faces 文件) 中的组件与 PageHandler 中的数据区和函数绑定。尽管 EGL 是不区分大小写的, 但 JSP 文件中引用的 EGL 名称的大小写必须与 EGL 变量或函数声明的大小写相同; 如果未能保持完全匹配, 将发生 JavaServer Faces 错误。建议在将该变量或函数与 JSP 字段绑定后不要更改 EGL 变量或函数的大小写。

有关命名问题的详细信息, 请参阅对 JSP 文件中的 EGL 标识和生成的 Java bean 的更改。

将组件与 PageHandler 中的数据区绑定

JSP 上的大多数组件都与数据具有一对一的相关性。例如, 文本框显示与该文本框绑定的 EGL 项的内容。如果用户更改了数据, 则输入文本框还将更新该 EGL 项。

当指定复选框组、列表框、单选按钮组或组合框时, 情况更为复杂。在那些情况下, 需要两种不同类型的绑定:

- 一种类型是将组件与要显示给用户的文本绑定。列表框中的项的文本是一个示例。
- 另一种类型是将组件与 PageHandler 数据区绑定, 该数据区接收值以指示用户的选择。例如, 您可以创建数据项以接收用户选择的列表框项的数字索引。

在“属性”视图中, 可以执行下列两个过程中的任何一个过程来将组件与用户见到的文本绑定:

- 可以使用**添加选项**来指示组件与单个字符串相关联, 可以显式地指定该字符串, 也可以通过标识 PageHandler 项来指定该字符串
- 可以使用**添加选项集合**来指示组件与字符串列表相关联, 可以显式地指定字符串列表, 也可以通过标识 PageHandler 区域 (如数据表或一组字符项) 来指定字符串列表

另外, 可以通过将一组字符项从“页数据”视图拖至该组件以将单选组件 (组合框、单选列表框或单选按钮组) 与该组字符项绑定。

要将组件与接收指示用户的选择的值的的数据区绑定, 可以使用“页数据”视图或“属性”视图来进行。该过程与绑定任何组件 (甚至是简单的文本框) 时的过程相同。

如果值只能是两个备用项中的一个, 则可以将该组件与 EGL 项绑定, 该项的项属性 **boolean** 设置为 *yes*。该组件用下列其中一个值来填充该项:

- 对于字符项, 该值是 **Y** (表示 *yes*) 或 **N** (表示 *no*)
- 对于数字项, 该值是 **1** (表示 *yes*) 或 **0** (表示 *no*)

当复选框显示时, 状态 (是否已选中) 视绑定项中的值而定。

有关可以应用于 PageHandler 中的数据项的属性的详细信息, 请参阅**页项属性**。

将组件与函数绑定

在将命令按钮或命令超链接拖放到页面上之后, 可以将该组件与现有 EGL 函数或 Page Designer 创建的事件处理程序绑定:

- 可以通过下列任何一种方式来将组件与现有事件处理程序绑定：
 - 通过将 EGL 函数从“页数据”视图中的“操作”节点拖至组件（按照建议）
 - 通过在“快速编辑”视图中打开组件
 - 通过右键单击组件并选择**编辑 Faces 命令事件**
- 当在“快速编辑”视图中打开组件或者右键单击组件并选择**编辑 Faces 命令事件**时，可以让 Page Designer 创建新的事件处理程序

如果 Page Designer 在 PageHandler 中创建了事件处理程序并允许您访问该 PageHandler 函数，则函数名就是工具指定的按钮标识加上字符串“Action”。如果该名称对 PageHandler 而言不是唯一的，则 Page Designer 将对函数名追加一个数字。

相关概念

『 PageHandler 』

相关任务

第 181 页的『创建 EGL 字段并将其与 Faces JSP 相关联』

第 182 页的『将 EGL 记录与 Faces JSP 相关联』

第 184 页的『对 PageHandler 代码使用“快速编辑”视图』

相关参考

第 619 页的『EGL 源格式的 PageHandler 部件』

第 624 页的『PageHandler 字段属性』

PageHandler

EGL *PageHandler* 是页面代码的一个示例；它控制用户与 Web 页面的运行时交互，并且可以执行下列任何任务：

- 将用于提交的数据值指定给 JSP 文件。这些值最终会显示在 Web 页面上。
- 更改从用户处或从被调用程序返回的数据。
- 将控制权转交给另一个 JSP 文件。

最简单的工作方式是在 Page Designer 中定制 JSP 文件并创建 PageHandler；有关详细信息，请参阅 EGL 的 *Page Designer* 支持。

PageHandler 本身包含变量和下列类型的逻辑：

- 一个 OnPageLoad 函数，JSP 第一次显示 Web 页面时将调用此函数
- 一组事件处理程序函数，作为对特定用户操作（明确地说，用户单击按钮或超文本链接）的响应，将调用这些事件处理程序函数中的每一个事件处理程序函数
- （可选）用来验证 Web 页面输入字段的验证函数
- 只能由其它 PageHandler 函数调用的专用函数

以两种方式访问 PageHandler 中的变量：

- 运行时环境自动访问数据。如果 JSP 中的字段与 PageHandler 中的项绑定，则结果如下所示：
 - 在 OnPageLoad 函数运行之后并且在 Web 页面显示之前，每个 PageHandler 项值都被写至与数据绑定的 JSP 字段。

- 当用户提交包含绑定的 JSP 字段的表单时，所提交的表单的每个字段中的值都将被复制到相关联的 PageHandler 项。仅在那时，才将控制权传递至事件处理程序。（但是，此描述未包括验证步骤，本主题随后的内容将讨论那些步骤。）
- 事件处理程序和 OnPageLoad 函数也可以与数据进行交互，并可以与数据存储（如 SQL 数据库）以及与被调用程序进行交互。

pageHandler 部件应该是简单的。虽然此部件可以包含轻量级数据验证（如范围检查），但建议您调用其它程序来执行复杂的业务逻辑。例如，应该让被调用程序来进行数据库访问。

与 PageHandler 相关联的输出

保存 PageHandler 时，EGL 会将 JSP 文件放在项目文件夹 WebContent\WEB-INF 中，但仅在以下情况下如此：

- 对 PageHandler **view** 属性指定了一个值，该值指定 JSP 文件名
- 文件夹 WebContent\WEB-INF 未包含具有指定名称的 JSP 文件

EGL 决不会覆盖 JSP 文件。

如果工作台首选项设置为在保存时自动进行构建，则每当您保存 PageHandler 时都会生成 PageHandler。在任何情况下，当生成 PageHandler 时，输出都由下列对象组成：

- 页 *bean* 是一个 Java 类，它包含数据并为 Web 页面提供初始化、数据验证和事件处理服务。
- 将在项目中的 JSF 配置文件中放置一个 <managed-bean> 元素，以便在运行时标识页 bean。
- 将在 JSF 应用程序配置文件中创建一个 <navigation-rule> 元素，以使 JSF 结果（PageHandler 的名称）与将要调用的 JSP 文件相关联。
- JSP 文件，在保存 PageHandler 时所处的情况下。

还将生成由部件处理程序使用的所有数据表和记录。

验证

如果基于 JSP 的 JSF 标记执行数据转换、验证或事件处理，则当用户提交 Web 页面时，JSF 运行时将立即执行必需的处理。如果找到错误，则 JSF 运行时可能重新显示页面，而不将控制权传递至 PageHandler。但是，如果 PageHandler 接收控制权，则 PageHandler 可能会执行一组基于 EGL 的验证。

如果在声明 PageHandler 时指定下列详细信息，则会进行基于 EGL 的验证：

- 各个输入字段的元素编辑（如最小输入长度）。
- 各个字段的基于类型的编辑（字符和数字）。
- 各个输入字段的 DataTable 编辑（范围、匹配有效和匹配无效），*DataTable* 部件对此作了说明。
- 各个输入字段的编辑功能。
- 整个 PageHandler 的编辑功能。

PageHandler 按以下顺序监视编辑，但仅对被用户更改了值的项执行监视：

1. 所有基本编辑和基于类型的编辑，即使某些编辑失败
2. （如果先前编辑成功的话）所有表编辑，即使某些编辑失败

3. （如果先前编辑成功的话）所有字段编辑功能，即使某些编辑功能失败
4. （如果所有先前编辑都成功的话）pageHandler 编辑功能

页项属性 **validationOrder** 定义各个输入字段的编辑顺序以及字段验证器函数的调用顺序。

如果未指定 validationOrder 属性，则缺省顺序是 PageHandler 中定义的项顺序，即从上到下。如果对 PageHandler 中的某些项（但不是所有项）定义了 validationOrder，则首先对所有带有 validationOrder 属性的项进行验证，并且是按指定的顺序进行的。然后，对不带 validationOrder 属性的项进行验证，按照该项在 PageHandler 中的顺序从上到下进行。

运行时方案

本节提供有关用户与 Web 应用程序服务器之间进行的运行时交互的技术概述。

当用户调用受 PageHandler 支持的 JSP 时，将执行下列步骤：

1. Web 应用程序服务器初始化环境：
 - a. 构造一个会话对象以便跨多次交互保留用户访问的应用程序所需的数据
 - b. 构造一个请求对象以便保留有关用户的当前交互的数据
 - c. 调用 JSP
2. JSP 处理一系列 JSF 标记以构造 Web 页面：
 - a. 创建一个 PageHandler 实例，致使使用用户指定的自变量来调用 onPageLoad 函数（如果有的话）并将 PageHandler 放到请求对象中
 - b. 访问存储在请求和会话对象中的数据，以将那些数据包括在 Web 页面中

注：pageHandler 部件有一个名为 onPageLoadFunction 的属性，该属性标识在 JSP 启动时调用的 PageHandler 函数。该函数自动检索任何传递给它的由用户提供的自变量；它可以调用其它代码；它可以将其它数据放到 Web 应用程序服务器的请求或会话对象中；但该函数既不会将控制权转交给另一个页面也不会第一次将页面显示给用户时显示错误消息。

3. JSP 将 Web 页面提交给用户，Web 应用程序服务器将破坏响应对象，而留下会话对象和 JSP。

如果用户在与 HTML <FORM> 标记相关联的屏幕字段中提供数据并提交表单，则执行下列步骤：

1. Web 应用程序服务器重新初始化环境：
 - a. 构造一个请求对象
 - b. 将对提交的表单接收到的数据放入页 bean 以进行验证
 - c. 重新调用 JSP
2. JSP 处理一系列 JSF 标记以将接收到的数据存储在页 bean 中。
3. 运行时 PageHandler 验证数据：
 - a. 执行相对基本的编辑（如最小输入长度），这些编辑是在 pageHandler 数据声明中指定的
 - b. 调用任何特定于项的验证函数，这些函数是在 pageHandler 数据声明中指定的
 - c. 调用 pageHandler 验证器函数，如果您希望根据另一个字段的内容来至少部分地验证一个字段，则需要这样做

(有关验证的详细信息, 请参阅上一节。)

4. 如果发生错误, 则 EGL 运行时将错误放在 JSF 队列中, 并且 JSP 重新显示带有嵌入消息的 Web 页面。但是, 如果未发生错误, 则结果如下所示:
 - a. 将存储在页 bean 中的数据写至记录 bean
 - b. 后续处理由事件处理程序确定, 该事件处理程序是在与用户单击的按钮或超链接相关联的 JSF 标记中标识的。

事件处理程序可以将处理转发至 JSF 标签, 该标签标识基于运行时 JSF 的配置文件中的映射。而映射又标识要调用的对象, 该对象是 JSP (通常是与 EGL PageHandler 相关联的 JSP) 或 servlet。

相关概念

第 20 页的『对部件的引用』

第 171 页的『Web 支持』

相关参考

第 175 页的『EGL 的 Page Designer 支持』

第 619 页的『EGL 源格式的 PageHandler 部件』

第 624 页的『PageHandler 字段属性』

JavaServer Faces 控件和 EGL

JavaServer Faces (JSF) 是一种服务器端用户界面组件框架。简单的说, JSF 是允许您为 Web 页面创建界面的一组工具和组件。JSF 组件可以在 Web 页面上显示数据并且接受用户的输入。

本主题说明 JSF 组件与 EGL 之间的关系。有关 JSF 的更多详细信息, 请参阅创建 Faces 应用程序 - 概述。要查看相关教程, 单击[帮助 > 教程库](#); 展开[边做边学](#); 然后选择使用 *JavaServer Faces* 在 Web 页面上显示动态信息。

有两种方法可用来利用 JSF 控件在 Web 页面上显示 EGL 数据:

- 可自动从“页数据”视图中的数据项或在 Page Designer 视图中创建的数据项创建 JSF 控件。要使用此方法, 在遵循将 EGL 记录与 Faces JSP 相关联或创建 EGL 数据项并将其与 Faces JSP 相关联中的指示时, 选择名为添加用于在 Web 页面上显示 EGL 元素的控件的复选框。
- 可手工添加 JSF 控件并将它们与“页数据”视图中的数据绑定。此方法允许您定制页面上的 JSF 控件布局, 而不是使用缺省布局。要使用此方法, 请参阅下列其中一个主题:
 - 将 *JavaServer Faces* 输入或输出组件与 *EGL PageHandler* 绑定
 - 将 *JavaServer Faces* 复选框组件与 *EGL PageHandler* 绑定
 - 将 *JavaServer Faces* 单选组件与 *EGL PageHandler* 绑定
 - 将 *JavaServer Faces* 多选组件与 *EGL PageHandler* 绑定

还可以将 PageHandler 中的 EGL 函数与 JSF 控件绑定。请参阅将 *JavaServer Faces* 命令组件与 *EGL PageHandler* 绑定。

相关任务

第 181 页的『创建 EGL 字段并将其与 Faces JSP 相关联』

第 182 页的『将 EGL 记录与 Faces JSP 相关联』

- 第 183 页的『将 JavaServer Faces 命令组件与 EGL PageHandler 绑定』
- 第 184 页的『将 JavaServer Faces 输入或输出组件与 EGL PageHandler 绑定』
- 第 185 页的『将 JavaServer Faces 复选框组件与 EGL PageHandler 绑定』
- 第 186 页的『将 JavaServer Faces 单选组件与 EGL PageHandler 绑定』
- 第 187 页的『将 JavaServer Faces 多选组件与 EGL PageHandler 绑定』

相关参考

- 第 175 页的『EGL 的 Page Designer 支持』

创建 EGL 字段并将其与 Faces JSP 相关联

要创建 EGL 基本字段并将其与 Faces JSP 相关联，执行下列操作：

1. 在 Page Designer 中打开 Faces JSP 文件。要打开 JSP 文件，在“项目资源管理器”中双击该 JSP 文件。该 JSP 文件便在 Page Designer 中打开了。单击设计选项卡以访问“设计”视图。

注：可通过在“设计”视图（或“源代码”视图）中单击鼠标右键并单击编辑页面代码来访问相关 PageHandler。

2. 从窗口菜单中，选择显示视图 > 其它 > 基本 > 选用板。
3. 在“选用板”视图中，单击 **EGL** 抽屉以显示 EGL 数据对象类型。
4. 将新建字段从选用板拖放到 JSP。这将显示“创建新的 EGL 数据字段”对话框。
5. 在名称字段中输入字段名称。
6. 从类型下拉列表中选择字段类型，并且如果需要指定字段的基本特征（长度和可能的小数位），则在维文本框中输入信息。如果声明下列类型的项，将使用缺省掩码：
 - 日期（掩码 *yyyymmdd*）
 - 时间（掩码 *hhmmss*）
 - 时间戳记（掩码 *yyyymmddhhmmss*）

如果希望将 DataItem 部件指定为类型，则选择 **DataItem**，它是列表中的最后一个值。在此情况下，将显示“选择 DataItem 部件”对话框，您可以从列表中选择一个 DataItem 部件，也可以输入名称，然后单击确定。

7. 如果要创建数据项数组，选择数组复选框，然后在大小文本框中输入整数。
8. 如果不想在页面中包括该字段，清除名为添加用于在 Web 页面上显示 EGL 元素的控件复选框并单击确定。该字段现在在“页数据”视图中。稍后可通过将该字段从“页数据”视图拖至 JSP 以在 JSP 文件中添加该字段。
9. 如果想要在 JSP 文件中包括这些字段，遵循下列附加步骤：
10. 选择名为添加用于在 Web 页面上显示 EGL 元素的控件复选框并单击确定。“插入控件”窗口将打开。
11. 在“插入控件”窗口时，选择指示您打算如何使用该字段的单选按钮：
 - 用于输出（显示现有记录）
 - 用于输入或输出（更新现有记录）
 - 用于输入（创建新记录）

您的选择将影响可用的控件类型。

12. 要更改字段标签，选择字段名旁边显示的标签，然后输入新内容。
13. 要选择标识类型之外的控件类型，从**控件类型**列表中选择类型。
14. 如果单击**选项**，将显示“选项”对话框，并且可用的特定选项取决于您要将该字段用于输入、输出还是同时用于两者。任何情况下的一个选项是在字段标签周围包括或排除 JSF 标记 `<h:outputLabel>`。

完成“选项”对话框后，单击**确定**。

15. 单击**完成**。

相关参考

第 175 页的『EGL 的 Page Designer 支持』

第 31 页的『基本类型』

将 EGL 记录与 Faces JSP 相关联

要将 EGL 记录与 Faces JSP 相关联，执行下列操作：

1. 在 Page Designer 中打开 Faces JSP 文件。如果尚未打开该 JSP 文件，则在“项目资源管理器”中双击该 JSP 文件。该 JSP 文件便在 Page Designer 中打开了。单击**设计**选项卡以访问“设计”视图。

注：可通过在“设计”视图（或“源代码”视图）中单击鼠标右键并单击**编辑页面代码**来访问相关 PageHandler。

2. 从**窗口**菜单中，选择**显示视图 > 其它 > 基本 > 选用板**。
3. 在“选用板”视图中，单击 **EGL** 抽屉以显示 EGL 数据对象类型。
4. 将**记录**从选用板拖放到 JSP 页面。这就显示了“选择记录部件”对话框。
5. 从列表中选择记录。
6. 指定字段名或接受缺省值，它是记录部件名。
7. 如果要声明记录数组，选择**数组**复选框，然后在**大小**文本框中输入整数。
8. 如果不想在页面中包括该记录，清除名为**添加用于在 Web 页面上显示 EGL 元素的控件**复选框并单击**确定**。该记录现在在“页数据”视图中。稍后可通过将该字段从“页数据”视图拖至 JSP 以在 JSP 文件中添加该字段。
9. 如果想要在 JSP 文件中包括这些字段，遵循下列附加步骤：
10. 选择对应**添加用于在 Web 页面上显示 EGL 元素的控件**的复选框并单击**确定**。“**插入控件**”窗口将打开。
11. 在“插入控件”窗口时，选择指示您打算如何使用该字段的单选按钮：
 - 用于输出（**显示现有记录**）
 - 用于输入或输出（**更新现有记录**）
 - 用于输入（**创建新记录**）

您的选择将影响可用的控件类型。

12. 要更改字段的顺序，使用**向上**和**向下**按钮。
13. 如果希望只选择一部分列示字段，单击**无**并选择需要的字段。要改为选择所有字段，单击**全部**。
14. 对每个字段执行下列操作：
 - a. 要排除字段，清除相关的复选框。要包括该字段，确保该复选框已选中。

- b. 要更改字段标签，选择字段名旁边显示的标签，然后输入新内容。
 - c. 要选择标识类型之外的控件类型（如果可能的话），从类型列表中进行选择。
15. 如果单击**选项**，将显示“选项”对话框，并且可用的特定选项取决于您要将该字段用于输入、输出还是同时用于两者。任何情况下的一个选项是在字段标签周围包括或排除 JSF 标记 `<h:outputLabel>`。

完成“选项”对话框后，单击**确定**。

16. 单击**完成**。

相关概念

第 122 页的『记录部件』

相关参考

第 175 页的『EGL 的 Page Designer 支持』

将 JavaServer Faces 命令组件与 EGL PageHandler 绑定

要将 JavaServer Faces 命令组件（按钮或超文本链接）与 EGL PageHandler 函数绑定，执行下列功能：

1. 在 Page Designer 中打开 Faces JSP 文件。如果尚未打开该 JSP 文件，则在“项目资源管理器”中双击该 JSP 文件。该 JSP 文件便在 Page Designer 中打开了。单击**设计选项卡**以访问“设计”视图。
 2. 从**窗口**菜单中，选择**显示视图 > 其它 > 基本 > 选用板**。
 3. 在“选用板”视图中，单击 **Faces 组件** 抽屉以显示 Faces 组件对象类型。
 4. 将命令组件从选用板拖至 JSP。命令组件的标号中具有**命令**字样。组件对象将放在 JSP 上。
 5. 使用下列其中一个方法将事件处理程序与命令组件绑定：
 - 要将组件与现有事件处理程序绑定，将事件处理程序从“页数据”视图中的“操作”节点拖至 JSP 上的组件对象
 - 要创建与组件绑定的新事件处理程序：
 - a. 右键单击该组件并从弹出菜单中单击**编辑事件**。
 - b. 使用“快速编辑”视图来输入用于事件处理程序的 EGL 代码。有关使用“快速编辑”视图的详细信息，请参阅对 *PageHandler* 代码使用“快速编辑”视图。
- “页数据”视图将显示该事件处理程序，并且对应的函数将被添加至 PageHandler。有关详细信息，请参阅 *EGL 源格式的 PageHandler 部件*。

相关概念

第 177 页的『PageHandler』

相关任务

第 175 页的『创建 EGL pageHandler 部件』

第 184 页的『对 PageHandler 代码使用“快速编辑”视图』

相关参考

第 175 页的『EGL 的 Page Designer 支持』

第 619 页的『EGL 源格式的 PageHandler 部件』

对 PageHandler 代码使用“快速编辑”视图

“快速编辑”视图允许您在不打开 PageHandler 文件的情况下维护 JSP 服务器事件的 EGL PageHandler 代码。要使用“快速编辑”视图，执行下列操作：

1. 在 Page Designer 中打开 JSP 文件。如果尚未打开该文件，则在“项目资源管理器”中双击该 JSP 文件。该 JSP 文件便在 Page Designer 中打开了。单击设计选项卡以访问“设计”视图。
2. 在 Page Designer 中单击鼠标右键，然后选择编辑事件。这就打开了“快速编辑”视图。
3. 执行下列步骤来维护命令组件的 PageHandler 函数：
 - a. 在 JSP 中选择命令组件。
 - b. 如果该命令组件已经具有与其相关联的 PageHandler 函数，则该函数将显示在“快速编辑”视图的脚本编辑器（右边窗格）中。对代码进行的任何更改都将反映在 PageHandler 中。
 - c. 要为所选命令组件创建 PageHandler 函数，请单击“快速编辑”视图的事件窗格中（左边窗格）中的命令，然后在“快速编辑”视图的脚本编辑器（右边窗格）中单击。将显示该函数。输入该函数的 PageHandler 代码。
4. 执行下列步骤来维护 onPageLoad 函数：
 - a. 在 JSP 中单击。
 - b. 在“快速编辑”视图的事件窗格（左边窗格）中单击 **onPageLoad**。
 - c. onPageLoad 函数将显示在“快速编辑”视图的脚本编辑器（右边窗格）中。对代码进行的任何更改都将反映在 PageHandler 中。

相关概念

第 177 页的『PageHandler』

相关参考

第 619 页的『EGL 源格式的 PageHandler 部件』

将 JavaServer Faces 输入或输出组件与 EGL PageHandler 绑定

要将 JavaServer Faces 输入或输出组件与现有 EGL PageHandler 数据区绑定，执行下列操作：

1. 在 Page Designer 中打开 Faces JSP 文件。如果尚未打开该 JSP 文件，则在“项目资源管理器”中双击该 JSP 文件。该 JSP 文件便在 Page Designer 中打开了。单击设计选项卡以访问“设计”视图。
2. 从窗口菜单中，选择显示视图 > 其它 > 基本 > 选用板。
3. 在“选用板”视图中，单击 **Faces 组件** 抽屉以显示 Faces 组件对象类型。
4. 将输入或输出组件从选用板拖至 JSP。输入和输出组件的标号中有输入和输出字样。组件对象将放在 JSP 上。
5. 要将该组件与现有 PageHandler 数据区绑定，执行下列其中一项操作：
 - 将数据区从“页数据”视图拖至 JSP 上的组件对象。
 - 在 JSP 上选择该组件对象，然后在“页数据”视图中右键单击该数据区并选择与“组件名”绑定。

- 在 JSP 上选择该组件对象。单击“属性”视图的**值**字段旁边的按钮，然后从“选择页数据对象”列表中选择数据区并单击**确定**。

相关概念

第 177 页的『PageHandler』

相关任务

第 175 页的『创建 EGL pageHandler 部件』

相关参考

第 175 页的『EGL 的 Page Designer 支持』

第 619 页的『EGL 源格式的 PageHandler 部件』

将 JavaServer Faces 复选框组件与 EGL PageHandler 绑定

JavaServer Faces 复选框组件是唯一的，这是因为与其绑定数据区的项属性 **isBoolean**（以前是 **boolean** 属性）必须设置为 **yes**。布尔数据区声明的示例如下所示：

```
DataItem CharacterBooleanItem char(1)
{
  value = "N",
  isBoolean = yes
}
end
DataItem NumericBooleanItem smallInt
{
  value = "0",
  isBoolean = yes
}
end
```

要将 JavaServer Faces 复选框组件与现有 EGL PageHandler 数据区绑定，执行下列操作：

1. 在 Page Designer 中打开 Faces JSP 文件。如果尚未打开该 JSP 文件，则在“项目资源管理器”中双击该 JSP 文件。该 JSP 文件便在 Page Designer 中打开了。单击**设计**选项卡以访问“设计”视图。
2. 从**窗口**菜单中，选择**显示视图 > 其它 > 基本 > 选用板**。
3. 在“选用板”视图中，单击 **Faces 组件** 抽屉以显示 Faces 组件对象类型。
4. 将复选框组件从选用板拖至 JSP。组件对象将放在 JSP 上。
5. 要将该组件与现有 PageHandler 数据区绑定，执行下列其中一项操作：
 - 将数据区从“页数据”视图拖至 JSP 上的组件对象。
 - 在 JSP 上选择该组件对象，然后在“页数据”视图中右键单击该数据区并选择与“**组件名**”绑定。
 - 在 JSP 上选择该组件对象。单击“属性”视图中的**值**字段旁边的按钮，然后从“选择页数据对象”列表中选择数据区并单击**确定**。

相关概念

第 177 页的『PageHandler』

相关任务

第 175 页的『创建 EGL pageHandler 部件』

相关参考

第 175 页的『EGL 的 Page Designer 支持』

第 619 页的『EGL 源格式的 PageHandler 部件』

将 JavaServer Faces 单选组件与 EGL PageHandler 绑定

单选组件允许用户从一系列值中选择一项。用户的选择将存储在 PageHandler 数据区中。单选按钮、单选列表框和组合框都是单选 JavaServer Faces 组件。

绑定是组件与数据区之间的一种关系。必须先在 PageHandler 中声明数据区，才能将其与组件绑定。单选组件需要两种不同类型的绑定：

- 与一个或多个数据区的绑定，这些数据区包含用户可从中进行选择的值
- 与将接收用户选择的数据区的绑定

要将 JavaServer Faces 单选组件与现有 EGL PageHandler 数据区绑定，执行下列操作：

1. 在 Page Designer 中打开 Faces JSP 文件。如果尚未打开该 JSP 文件，则在“项目资源管理器”中双击该 JSP 文件。该 JSP 文件便在 Page Designer 中打开了。单击设计选项卡以访问“设计”视图。
2. 从窗口菜单中，选择显示视图 > 其它 > 基本 > 选用板。
3. 在“选用板”视图中，单击 **Faces 组件** 抽屉以显示 Faces 组件对象类型。
4. 将单选组件从选用板拖至 JSP。组件对象将放在 JSP 上。
5. 要将组件与一个或多个 PageHandler 数据区绑定，而这些数据区包含要显示给用户的值，则执行下列其中一个过程：
 - 可将组件与各个 PageHandler 数据区绑定，每个数据区包含一个列表项。对每个数据区执行以下过程：
 - a. 在 JSP 上选择该对象组件。
 - b. 在“属性”视图中，单击**添加选项**。“名称”和“值”字段中填充了缺省值。
 - c. 单击**名称**字段，然后输入要显示给用户的文本。
 - d. 单击**值**字段，然后单击**值**字段旁边的按钮。从“选择页数据对象”列表中选择单个数据区并单击**确定**。此数据区包含将移至接收数据区的值。
 - 可将组件与 PageHandler 数组数据区绑定，该数据区包含要显示给用户的值。执行以下过程以将该组件与数组数据区绑定：
 - a. 在 JSP 上选择该对象组件。
 - b. 在“属性”视图中，单击**添加选项组**。“名称”和“值”字段中填充了缺省值。
 - c. 单击**值**字段，然后单击**值**字段旁边的按钮。从“页数据对象”列表中选择数组数据区并单击**确定**。数组数据区中的值就是要显示给用户的值。稍后描述的属性将确定移至接收数据区的是数组数据区中的值还是功能相当的索引值。
6. 如果要使用数组数据区来提供向用户显示的值，必须使用以下两个属性来定义接收数据区：**selectFromListItem** 和 **selectType**。selectFromListItem 属性指向包含列表项的数组。selectType 属性指示是以文本值还是以索引值来填充接收数据区。以下是接收数据区的一些示例：

```
colorSelected char(10)
{selectFromListItem = "colorsArray",
 selectType = value};
```

```
colorSelectIdx smallInt
{selectFromListItem = "colorsArray",
selectType = index};
```

7. 要将该组件与将接收用户选择的 PageHandler 数据区绑定，执行下列其中一项操作：
 - 将数据区从“页数据”视图拖至 JSP 上的组件对象。
 - 在 JSP 上选择该组件对象，然后在“页数据”视图中右键单击该数据区并选择与“组件名”绑定。
 - 在 JSP 上选择该组件对象。单击“属性”视图中的值字段旁边的按钮，然后从“选择页数据对象”列表中选择数据区并单击确定。

相关概念

第 177 页的『PageHandler』

相关任务

第 175 页的『创建 EGL pageHandler 部件』

相关参考

第 175 页的『EGL 的 Page Designer 支持』

第 619 页的『EGL 源格式的 PageHandler 部件』

将 JavaServer Faces 多选组件与 EGL PageHandler 绑定

多选组件允许用户从一系列值中选择一项或多项。用户的选择将存储在 PageHandler 数组数据区中。复选框组和多选列表框都是多选 JavaServer Faces 组件。

绑定是组件与数据区之间的一种关系。必须先在 PageHandler 中声明数据区，才能将其与组件绑定。多选组件需要两种不同类型的绑定：

- 与一个或多个数据区的绑定，这些数据区包含用户可从中进行选择的值
- 与将接收用户选择的数组数据区的绑定

要将 JavaServer Faces 多选组件与现有 EGL PageHandler 数据区绑定，执行下列操作：

1. 在 Page Designer 中打开 Faces JSP 文件。如果尚未打开该 JSP 文件，则在“项目资源管理器”中双击该 JSP 文件。该 JSP 文件便在 Page Designer 中打开了。单击设计选项卡以访问“设计”视图。
2. 从窗口菜单中，选择显示视图 > 其它 > 基本 > 选用板。
3. 在“选用板”视图中，单击 **Faces 组件** 抽屉以显示 Faces 组件对象类型。
4. 将多选组件从选用板拖至 JSP。组件对象将放在 JSP 上。
5. 要将组件与一个或多个 PageHandler 数据区绑定，而这些数据区包含要显示给用户的值，则执行下列其中一个过程：
 - 可将组件与各个 PageHandler 数据区绑定，每个数据区包含一个列表项。对每个数据区执行以下过程：
 - a. 在 JSP 上选择该对象组件。
 - b. 在“属性”视图中，单击添加选项。“名称”和“值”字段中填充了缺省值。
 - c. 单击名称字段，然后输入要显示给用户的文本。
 - d. 单击值字段，然后单击值字段旁边的按钮。从“选择页数据对象”列表中选择单个数据区并单击确定。此数据区包含将移至接收数据区的值。

- 可将组件与 PageHandler 数组数据区绑定，该数据区包含要显示给用户的值。执行以下过程以将该组件与数组数据区绑定：
 - a. 在 JSP 上选择该对象组件。
 - b. 在“属性”视图中，单击**添加选项组**。“名称”和“值”字段中填充了缺省值。
 - c. 单击**值**字段，然后单击**值**字段旁边的按钮。从“选择页数据对象”列表中选择数组数据区并单击**确定**。数组数据区中的值就是要显示给用户的值。稍后描述的属性将确定移至接收数据区的是数组数据区中的值还是功能相当的索引值。
- 6. 如果要使用数组数据区来提供向用户显示的值，必须使用以下两个属性来定义接收数据区：**selectFromListItem** 和 **selectType**。selectFromListItem 属性指向包含列表项的数组。selectType 属性指示是以文本值还是以索引值来填充接收数据区。以下是接收数据区的一些示例：

```
colorSelected char(10)
{selectFromListItem = "colorsArray",
 selectType = value};

colorSelectIdx smallInt
{selectFromListItem = "colorsArray",
 selectType = index};
```
- 7. 要将该组件与接收用户选择的 PageHandler 数组数据区绑定，执行下列操作：
 - a. 在 JSP 上选择该组件对象
 - b. 单击“属性”视图的**值**字段旁边的按钮
 - c. 从“选择页数据对象”列表中选择数据区
 - d. 单击**确定**

相关概念

第 177 页的『PageHandler』

相关任务

第 175 页的『创建 EGL pageHandler 部件』

相关参考

第 175 页的『EGL 的 Page Designer 支持』

第 619 页的『EGL 源格式的 PageHandler 部件』

创建 EGL 报告

EGL 报告概述

EGL 可以根据 JasperReports 的功能生成报告，JasperReports 是基于 Java 的开放式源代码报告库。有关该库的详细信息，请参阅以下 Web 站点：

<http://jasperreports.sourceforge.net>

EGL 不提供报告布局的机制。必须执行下列操作：

- 导入 JasperReports 输出文件（文件扩展名为 *jasper*）；或者
- 使用文本编辑器或专业工具来创建设计文件，在“工作台”中单击项目 > 全部构建时该文件将变换为 JasperReports 输出文件。

下面是用于创建设计文件的两个专业工具：

- JasperAssistant，如以下 Web 站点中所述：

<http://www.jasperassistant.com>

- iReport，如以下 Web 站点中所述：

<http://ireport.sourceforge.net>

在写至驱动报告生产的 EGL 文件中，先将数据提交至 JasperReports 输出文件（或接受在该文件中指定的数据源），然后将报告导出至一个或多个输出文件，每个文件可能为不同格式，如 HTML 或 Adobe Acrobat PDF。

如果同时编写类型为 JasperReport 的 EGL 处理程序，可以响应在使用数据填充报告时发生的用户事件；例如，可在报告生产快要完成时将特定于运行的详细信息添加至报告。但是，要确保事件处理起作用，必须确保在 JasperReports 输出文件中引用了从报告处理程序生成的输出。

EGL 报告处理程序向导允许您轻松创建 EGL 报告处理程序。

在编写用来与报告交互的 EGL 代码时，使用系统库 **ReportLib** 中的函数；在产生报告的代码中，创建类型为 Report 和 ReportData 的变量。

已经为您定义了此处提到的所有 EGL 部件（处理程序、报告和报告定义）。

相关概念

EGL 报告创建过程概述

相关参考

EGL 报告库

数据源

EGL 报告处理程序

EGL 报告创建过程概述

本主题概述为 EGL 项目创建和生成报告的一般过程。有关这些过程的其它详细信息包含在 EGL 报告任务帮助主题中。

要创建报告，应完成下面所述的三个过程。其中用于创建 XML 设计和编写用于生成报告的代码的两个过程是必需的。用于创建报告处理程序的第三个过程是可选的。您不一定要按描述的顺序完成这些过程。例如，如果需要，可以在创建 XML 设计文档之前创建报告处理程序，也可以同时创建它们，但在下面步骤 2 中的“报告处理程序与 XML 设计文档之间的代码相互关系”段落中描述的情况中除外。

如果没有 XML 设计文档以及用于生成报告的代码，则不能生成报告。

创建报告需要完成的三个过程包括：

1. 创建 XML 设计文档以指定报告的布局信息。可以下列任一方式来创建此文档：
 - 通过使用第三方 JasperReport 设计工具（如 Jasper Assistant 或 iReports）。
 - 通过使用文本编辑器来将 Jasper XML 设计信息编写到新的文本文件中。

XML 设计文档必须具有 .jrxml 扩展名。如果您创建的文件没有此扩展名，则将该文件重命名为 .jrxml 文件。此外，一定要将 XML 设计文档保存在将包含 EGL 报告处理程序和报告调用代码文件的 EGL 包中。

您创建的 .jrxml 文件将编译成 .jasper 文件。如果未创建新的 .jrxml 文件，则必须导入先前编译的 .jasper 文件。

2. 如果想要使用报告处理程序（该处理程序提供用于处理填写报告时发生的事件的逻辑），可以下列任一方式创建报告处理程序：
 - 通过使用 EGL 报告处理程序向导来指定报告处理程序的信息。
 - 通过创建新的 EGL 源文件并使用报告处理程序模板插入处理程序或者手工输入处理程序代码。

报告处理程序与 XML 设计文档之间的代码相互关系。在 .jrxml 文件中，可以指定 `scriptletClass`，它引用从 EGL 报告处理程序生成的报告处理程序文件。注意：

- 如果 .jrxml 文件使用报告处理程序产生的 Java 代码，则必须先生成报告处理程序才能创建 .jrxml 文件。
 - 如果更改报告处理程序，则必须重新编译 .jrxml 文件。
 - 如果需要解析 .jrxml 文件中的所有编译错误或者想要在更改报告处理程序后重新编译 .jasper 文件，则必须修改 .jrxml 文件并保存它。
3. 使用 EGL `ReportLib` 函数以在 EGL 项目中编写报告调用代码。可在创建报告调用代码时使用 EGL 程序部件向导。

要点：给定的报告处理程序和报告调用代码的文件名与 XML 设计文档的名称一定不能相同。如果不这么做，设计文件的编译会导致覆盖 Java 代码。为避免出现问题，将报告处理程序命名为 `reportName_handler.egl`，并将 XML 设计文档命名为 `reportName_XML.jrxml`。例如，可将报告命名为 `abc_handler.egl` 并将设计文档命名为 `abc_XML.jrxml`。您还必须确保 XML 设计文件具有唯一名称，不会与任何 EGL 程序文件冲突。

要在创建 XML 设计文档、报告处理程序（如果想要使用的话）和报告调用代码后构建并生成报告，应完成下列过程：

1. 通过选择项目 > 全部构建来构建 EGL 项目。

EGL 自动从 EGL 报告处理程序生成 Java 代码并将 XML 设计文档（.jrxml 文件）编译成 .jasper 文件。

2. 运行具有报告调用代码的 EGL 程序。

在 EGL 程序运行后，EGL 使用的 JasperReports 程序将生成的报告自动保存在报告调用代码中的 *reportDestinationFileName* 指定的位置。

生成报告的 JasperReports 程序还会生成 .jprint 文件并存储它，这是一种中间文件格式，它将导出至最终报告格式（.pdf、.html、.xml、.txt 或 .csv）。

程序可将一个 .jprint 重复用于多次导出。

报告调用代码中的 `exportReport()` 函数使 EGL 以指定格式导出报告。例如，以下代码使 EGL 以 .pdf 格式导出报告：

```
reportLib.exportReport(myReport, ExportFormat.pdf);
```

EGL 不会自动刷新导出的报告。如果更改报告设计或者数据更改，则必须重新填写报告并重新导出它。

相关概念

EGL 报告概述

相关任务

将设计文档添加至包

使用报告模板

创建 EGL 报告处理程序

手工创建 EGL 报告处理程序

编写用于生成报告的代码

运行报告

导出报告

使用 EGL 中的内容辅助

相关参考

EGL 报告库

数据源

EGL 报告处理程序

数据源

EGL 报告库包括对包含数据的主数据源的引用以及对获取数据方式的信息的引用。

可使用下列语句来指定数据源信息：

- *DataSource.databaseConnection*
- *DataSource.sqlStatement*
- *DataSource.reportData*

例如，如果在使用 `ReportLib.fillReport` 函数时指定 `fillReport (eglReport, DataSource.databaseConnection)`，EGL 将检索数据库连接并将其传递至 JasperReports 引擎。

有关如何将数据库连接、报告数据和 SQL 语句用作数据源以生成报告的示例，请参阅 EGL 报告驱动程序函数的样本代码。

相关概念

第 189 页的『EGL 报告概述』

第 190 页的『EGL 报告创建过程概述』

相关参考

第 788 页的『EGL 库 ReportLib』

第 791 页的『fillReport()』

第 197 页的『EGL 报告驱动程序函数的样本代码』

库中的数据记录

EGL 库包含 `ReportData` 记录和报告记录。

`ReportData` 记录包含有关要在报告中使用的特定数据的信息。该记录包含下列字段：

字段	说明	数据类型
<code>connectionName</code>	在 EGL 程序中建立的连接的名称。	字符串
<code>sqlStatement</code>	EGL 应该执行的 SQL 语句。来自语句的执行结果的报告数据。	字符串
<code>Data</code>	动态记录数组。	Any (这是 EGL Any 类型。)

报告记录包含有关特定报告的信息。该记录包含下列字段：

字段	说明	数据类型
<code>reportDesignFile</code>	报告设计文件的名称，该文件是带有 .jrxml 扩展名的 XML 文件。	字符串
<code>reportDestinationFile</code>	.jrprint 文件的位置。	字符串
<code>reportExportFile</code>	最终保存的 .xml、.pdf、.html、.txt 或 .csv 文件的位置。	字符串
<code>reportData</code>	用作报告的主数据源的报告数据。	

相关概念

第 189 页的『EGL 报告概述』

第 190 页的『EGL 报告创建过程概述』

相关参考

第 788 页的『EGL 库 ReportLib』

EGL 报告处理程序

EGL 报告处理程序提供附加功能以处理在用数据填写报告时发生的事件。可使用“新建 EGL 报告处理程序”向导来指定有关报告处理程序的信息，或者可以手工创建报告处理程序。

生成报告处理程序文件时，EGL 将创建下列文件：

- *handlerName.java*
- *handlerName_lib.java* 文件。

handlerName

EGL 报告处理程序的别名

在 EGL 生成 .java 文件时，类名是小写的。在 XML 设计文档中输入的任何类名都必须是小写的。

有关报告处理程序代码的样本代码和示例，请参阅手工创建 EGL 报告处理程序。

技术详细信息： EGL 报告处理程序是类型为 `JasperReport` 的 EGL 处理程序部件。报告处理程序映射至 `JasperReports` scriptlet 类。报告处理程序 Java 生成扩展了 `JRDefaultScriptlet` 类并定义了 Java 类，该 Java 类包含生成的用于表示 scriptlet 函数的 Java 函数。XML 设计文档的定义部分包含 scriptlet 类的名称。`JasperReports` 引擎装入 scriptlet 类并按报告定义中定义的那样调用不同的方法。（有关 `JasperReports` scriptlet 和 scriptlet 类的更多信息，请参阅 `JasperReports` 文档。）

报告处理程序维护请求时返回的 `ReportData` 记录的内部列表。

相关概念

第 189 页的『EGL 报告概述』

第 190 页的『EGL 报告创建过程概述』

相关任务

第 93 页的『将 EGL 代码迁移至 EGL 6.0 iFix』

第 201 页的『手工创建 EGL 报告处理程序』

第 205 页的『编写用于生成报告的代码』

相关参考

第 195 页的『其它 EGL 报告处理程序函数』

第 788 页的『EGL 库 ReportLib』

第 194 页的『预定义报告处理程序函数』

预定义报告处理程序函数

报告处理程序提供下列预定义函数，您可以将它们用作函数模板：

功能	函数运行位置
beforeReportInit();	报告初始化之前
afterReportInit();	报告初始化之后
beforePageInit();	进入页面
afterPageInit();	离开页面
beforeColumnInit();	列初始化之前
afterColumnInit();	列初始化之后
beforeGroupInit (<i>groupName</i> String);	组初始化之前。 <i>groupName</i> 是报告中该组的名称。
afterGroupInit(<i>groupName</i> String);	组初始化之后。
beforeDetailEval();	每个字段之前。如果设置了此函数，每一行在显示之前将调用此函数。
afterDetailEval();	每个字段之后。如果设置了此函数，每一行在显示之前将调用此函数。

在其中一个函数中，可调用其它函数。例如，可调用 `setReportVariable()`，如下所示：

```
function afterGroupInit(groupName String)
  if (groupName == "cat")
    setReportVariableValue ("NewGroupName", "dog");
  else
    setReportVariableValue ("NewGroupName", groupName);
  end
end
```

还可以创建您自己的函数。有关创建定制函数的信息，请参阅 [JasperReports 文档](#)。

有关使用预定义报告处理程序函数的示例，请参阅 [手工创建 EGL 报告处理程序](#)。

相关概念

第 189 页的『[EGL 报告概述](#)』

第 190 页的『[EGL 报告创建过程概述](#)』

相关任务

第 93 页的『[将 EGL 代码迁移至 EGL 6.0 iFix](#)』

第 201 页的『[手工创建 EGL 报告处理程序](#)』

相关参考

第 195 页的『[其它 EGL 报告处理程序函数](#)』

第 192 页的『[库中的数据记录](#)』

第 788 页的『[EGL 库 ReportLib](#)』

第 193 页的『[EGL 报告处理程序](#)』

其它 EGL 报告处理程序函数

可在预定义报告处理程序函数中调用下列任何 ReportLib 函数:

用于获取报告参数的函数

函数	用途
getReportParameter (<i>parameter</i> String <u>in</u>)	从要填写的报告返回指定参数的值。

用于设置和获取报告变量的函数

这些变量可以用在很多地方, 例如, 用于存储经常使用的表达式, 或者用于对表达式 (该表达式是在要处理的行上定义的) 定义复杂计算。

功能	用途
getReportVariableValue (<i>variable</i> String <u>in</u>)	从要填写的报告返回指定变量的值。返回的值为 ANY 类型。
setReportVariableValue (<i>variable</i> String <u>in</u> , <i>value</i> Any <u>in</u>);	将指定变量的值设置为提供的值。

获取字段值的函数

功能	用途
getFieldValue (<i>fieldName</i> String <u>in</u>)	对当前处理的行返回指定字段值。返回的值为 ANY 类型。

用于添加或获取有关子报告的数据的函数

子报告是从另一个报告中调用的报告。有时会在主报告与子报告之间交换数据。子报告还可以是另一个子报告的主报告。

功能	用途
addReportData (<i>rd</i> ReportData <u>in</u> , <i>dataSetName</i> String <u>in</u>);	将带有指定名称的报告数据对象添加至当前报告处理程序。
getReportData (<i>dataSetName</i> String <u>in</u>)	检索带有指定名称的报告数据记录。返回的值为 ReportData 类型。

有关使用此主题中描述的函数的示例, 请参阅[手工创建 EGL 报告处理程序](#)。

相关概念

第 190 页的『EGL 报告创建过程概述』

第 189 页的『EGL 报告概述』

相关任务

第 93 页的『将 EGL 代码迁移至 EGL 6.0 iFix』

第 201 页的『手工创建 EGL 报告处理程序』

相关参考

- 第 192 页的『库中的数据记录』
- 第 193 页的『EGL 报告处理程序』
- 第 788 页的『EGL 库 ReportLib』
- 第 194 页的『预定义报告处理程序函数』

XML 设计文档中的数据类型

在 XML 报告设计文档中，数据类型被描述为 Java 数据类型。如果创建要在设计文档中使用的 EGL scriptlet 代码，使用与适用 EGL 基本类型相对应的 Java 数据类型。必须使用 Java 数据类型声明因为调用 EGL scriptlet 代码而返回的数据。

下表显示 EGL 基本类型如何映射至 Java 数据类型。JavaReports 文档包含有关您可以使用的 Java 数据类型的信息。

EGL 基本类型	Java 数据类型
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

相关概念

- 第 189 页的『EGL 报告概述』
- 第 190 页的『EGL 报告创建过程概述』

相关任务

第 198 页的『将设计文档添加至包』

相关参考

第 192 页的『库中的数据记录』

第 788 页的『EGL 库 ReportLib』

第 193 页的『EGL 报告处理程序』

EGL 报告驱动程序函数的样本代码

此主题包含一些代码段，它们将显示如何使用三个不同数据源生成报告：

- 数据库连接
- 数据记录
- SQL 语句

以下代码段显示如何通过将数据库连接用作数据源来生成报告：

```
//Variable declaration
myReport    Report;
myReportData ReportData;

//Function containing report invocation code
function makeReport()
    //Initialize Report file locations
    myReport.reportDesignFile = "reportDesignFileName.jasper";
    myReport.reportDestinationFile =
"reportDestinationFileName.jrprint";

    //Set the report data via a connection using the SQL statement
    //embedded in the report design
    sysLib.defineDatabaseAlias("alias", "databaseName");
    sysLib.connect("alias", "userid", "password");
    myReportData.connectionName="connectionName";
    myReport.reportData = myReportData;

    //Fill the report with data
    reportLib.fillReport(myReport, DataSource.databaseConnection);
    //Export the report in PDF format
    myReport.reportExportFile = "reportDesignFileName.pdf";
    reportLib.exportReport(myReport, ExportFormat.pdf);
end
```

以下代码段显示如何通过将报告数据灵活记录用作数据源来生成报告：

```
//Variable declaration
myReport    Report;
myReportData ReportData;

//Function containing the report driving code
function makeReport()
    //Initialize myReport file locations
    myReport.reportDesignFile = "reportDesignFileName.jasper";
    myReport.reportDestinationFile =
"reportDestinationFileName.jrprint";

    //Set the report data
    populateReportData();
    myReport.reportData = myReportData;

    //Fill the report with data
```

```

reportLib.fillReport(myReport, DataSource.reportData);

//Export the report in HTML format
myReport.reportExportFile = "reportDesignFileName.html";
reportLib.exportReport(myReport, ExportFormat.html);
end

function populateReportData()
//Insert EGL code here which populates myReportData
...
end

```

以下代码段显示如何通过将 SQL 语句用作数据源来生成报告:

```

//Variable declaration
myReport Report;
myReportData ReportData;

//Function containing report driving code
function makeReport()
//Initialize Report file locations
myReport.reportDesignFile = "reportDesignFileName.jasper";
myReport.reportDestinationFile = "reportDestinationFileName.jrprint";

//Set the report data via a SQL statement
myReportData.sqlStatement = "SELECT * FROM dataBaseTable";
myReport.reportData = myReportData;

//Fill the report with data
reportLib.fillReport(myReport, DataSource.sqlStatement);

//Export the report in text format
myReport.reportExportFile = "reportOutputFileName.txt";
reportLib.exportReport(myReport, ExportFormat.text);
end

```

相关概念

第 190 页的『EGL 报告创建过程概述』

第 189 页的『EGL 报告概述』

相关任务

第 205 页的『编写用于生成报告的代码』

相关参考

第 192 页的『库中的数据记录』

第 191 页的『数据源』

第 193 页的『EGL 报告处理程序』

第 788 页的『EGL 库 ReportLib』

将设计文档添加至包

必须具有带 .jrxml 扩展名的 XML 设计文档，它会指定报告的布局和其它设计信息。

要将设计文档添加至包，遵循下列步骤:

1. 以下列任何方式创建设计文档:

- 使用第三方 JasperReports 设计工具（如 JasperAssistant 或 iReports）。如果创建的文件没有 .jrxml 扩展名，则重命名该文件以使该文件具有 .jrxml 扩展名。
- 使用文本编辑器以将 Jasper XML 设计信息编写到新的文本文件中，并将该文件另存为 .jrxml 文件。

2. 将 XML 设计文档放在将包含 EGL 报告处理程序和报告驱动程序文件的 EGL 包中。

如果未创建新的 XML 设计文档，则必须导入先前编译的 .jasper 文件。

当您选择 **项目 > 全部构建** 来构建所有 EGL 项目组件时，.jrxml 文件将自动编译成 .jasper 文件。

注：如果要同时创建 XML 设计文档和报告处理程序，请参阅 [EGL 报告创建过程概述](#) 以获取要遵循的准则。有关显示 XML 设计文档如何从报告处理程序获取报告数据记录的示例，请参阅 [手工创建 EGL 报告处理程序](#)。

相关概念

[EGL 报告概述](#)

[EGL 报告创建过程概述](#)

相关任务

[创建 EGL 报告处理程序](#)

[手工创建 EGL 报告处理程序](#)

[编写用于生成报告的代码](#)

相关参考

[EGL 报告库](#)

[XML 设计文档中的数据类型](#)

使用报告模板

可选择并修改下列任何 EGL 报告模板：

- 数据库连接模板
- 报告数据模板
- SQL 语句模板
- 报告处理程序模板

要使用报告模板，遵循下列步骤：

1. 选择 **窗口 > 首选项**。
2. 首选项列表显示后，展开 **EGL**。
3. 展开 **编辑器**，然后选择 **模板**。
4. 滚动模板列表并选择一个模板。例如，选择 **处理程序** 以显示报告处理程序模板。
5. 单击 **编辑**。
6. 更改模板以满足您的需要。

输入 **handler**，然后按 **Alt+/** 键以编辑报告处理程序模板。有关包括代码示例在内的更多信息，请参阅 [手工创建 EGL 报告处理程序](#)。

输入 **jas**，然后按 **Alt+/** 以编辑数据源模板。

7. 单击 **应用**，然后单击 **确定** 以保存更改。

相关概念

EGL 报告概述
EGL 报告创建过程概述

相关任务

手工创建 EGL 报告处理程序
编写用于生成报告的代码
使用 EGL 中的内容辅助
设置模板首选项

相关参考

EGL 报告处理程序
EGL 报告库
EGL 报告驱动程序函数的样本代码

创建 EGL 报告处理程序

EGL 报告处理程序提供用于处理在填写报告时发生的事件的逻辑。要创建 EGL 报告处理程序，执行下列操作：

1. 标识用来包含文件的项目或文件夹。如果还没有项目或文件夹，则必须创建项目或文件夹。
2. 在工作台中，单击**文件 > 新建 > 其它**。
3. 在“新建”窗口中，展开 **EGL**。
4. 单击**报告处理程序**。
5. 单击**下一步**。
6. 选择将包含 EGL 文件的项目或文件夹，然后选择包。
7. 由于报告处理程序名将与文件名完全相同，因此选择一个遵从 EGL 部件名约定的文件名。在“EGL 源文件名”字段中，输入 EGL 文件的名称，例如，myReportHandler。
8. 单击**完成**。

相关概念

第 7 页的『开发过程』
第 13 页的『EGL 项目、包和文件』
第 483 页的『生成的输出』
第 16 页的『部件』
第 8 页的『运行时配置』

相关任务

第 116 页的『创建 EGL Web 项目』

相关参考

第 441 页的『EGL 编辑器』
第 448 页的『EGL 源格式』

手工创建 EGL 报告处理程序

如果不想使用“新建 EGL 报告处理程序向导”来创建报告处理程序，可手工创建报告处理程序。

要手工创建报告处理程序，遵循下列步骤：

1. 创建新的 EGL 源文件。
2. 或者：
 - 手工输入处理程序代码。
 - 使用报告处理程序模板插入处理程序，如下所示：
 - a. 浏览至报告处理程序模板并选择想要的模板。
 - b. 单击**编辑**。
 - c. 输入 **handler**，然后按 **Alt+/** 键。
 - d. 根据需要更改模板代码。

本主题的余下部分包含显示下列各项的代码示例：

- 用于手工创建报告处理程序的语法
- 如何在报告处理程序中获取报告参数
- 如何设置和获取报告变量
- 如何获取字段值
- 如何添加灵活记录
- XML 设计文档如何从报告处理程序获取报告数据记录

可针对您的应用程序复制此代码并修改它。

样本代码，它将显示用于手工创建报告处理程序的语法

以下代码显示用于手工创建 EGL 报告处理程序的一般语法：

```
handler handlerName type jasperReport

// Use Declarations (optional)
use usePartReference;

// Constant Declarations (optional)
const constantName constantType = literal;

// Data Declarations (optional)
identifierName declarationType;

// Pre-defined Jasper callback functions (optional)
function beforeReportInit()
...
end

function afterReportInit()
...
end

function beforePageInit()
...
end

function afterPageInit()
...
end
```

```

end

function beforeColumnInit()
...
end

function afterColumnInit()
...
end

function beforeGroupInit(stringVariable string)
...
end

function afterGroupInit(stringVariable string)
...
end

function beforeDetailEval()
...
end

function afterDetailEval()
...
end

// User-defined functions (optional)
function myFirstFunction()
...
end

function mySecondFunction()
...
end
end

```

显示如何获取报告参数的示例

以下代码段显示如何在报告处理程序中获取报告参数:

```

handler myReportHandler type jasperReport

// Data Declarations
report_title String;

// Jasper callback function
function beforeReportInit()
...

    report_title = getReportTitle();

...
end

...

// User-defined function
function getReportTitle() Returns (String)
    return (getReportParameter("ReportTitle"));
end

end

```

显示如何设置和获取报告变量的示例

下面的代码段显示如何在报告处理程序中设置和获取报告变量:

```

handler myReportHandler type jasperReport

    // Data Declarations
    employee_serial_number int;

    // Jasper callback function
    function afterPageInit()
    ...
        employee_serial_number = getSerialNumberVar();
    ...
    end

    ...

    // User-defined function
    function getSerialNumberVar() Returns (int)
        employeeName String;
        employeeName = "Ficus, Joe";
        setReportVariableValue("employeeNameVar", employeeName);
        return (getReportVariableValue("employeeSerialNumVar"));
    end
end

```

显示如何在报告处理程序中获取字段值的示例

以下示例代码段显示如何在报告处理程序中获取报告字段值:

```

handler myReportHandler type jasperReport

    // Data Declarations
    employee_first_name String;

    // Jasper callback function
    function beforeColumnInit()
    ...
        employee_first_name = getFirstNameField();
    ...
    end

    ...

    // User-defined function
    function getFirstNameField() Returns (String)
        fldName String;
        fldName = "fname";
        return (getFieldValue(fldName));
    end

end

```

显示如何在报告处理程序中添加报告数据灵活记录的示例

下面的示例代码显示如何在报告处理程序中添加报告数据灵活记录:

```

handler myReportHandler type jasperReport

    // Data Declarations
    customer_array customerRecordType[];
    c    customerRecordType;

    // Jasper callback function
    function beforeReportInit()
        customer ReportData;
        datasetName String;

        //create the ReportData object for the Customer subreport
        c.customer_num = getFieldValue("c_customer_num");
    end
end

```

```

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.addElement(c);
customer.data = customer_array;
datasetName = "customer";
addReportData(customer, datasetName);
end
end

```

显示 XML 设计文档如何从报告处理程序获取报告数据记录的示例

下面的代码段显示 XML 设计文档如何从报告处理程序获取报告数据灵活记录:

```

<jasperReport name="MasterReport"
  scriptletClass="subreports.SubReportHandler">
  ...
  <subreport>
    <dataSourceExpression>
      <![CDATA[(JRDataSource)(((subreports.SubReportHandler)
        ${REPORT_SCRIPTLET}).getReportData(
          new String("customer")))]>&gt;
    </dataSourceExpression>
    <subreportExpression class="java.lang.String">
      <![CDATA["C:/RAD/workspaces/Customer.jasper"]>&gt;
    </subreportExpression>
  </subreport>
  ...
</jasperReport>

```

相关概念

EGL 报告概述

EGL 报告创建过程概述

相关任务

创建 EGL 源文件

创建 EGL 报告处理程序

使用报告模板

相关参考

EGL 报告库

EGL 报告处理程序

预定义报告处理程序函数

其它 EGL 报告处理程序函数

编写用于生成报告的代码

使用“新建 EGL 程序部件”向导来创建新的 EGL 基本程序，该程序将使用报告库来运行报告。

要创建报告驱动程序，遵循下列步骤：

1. 选择文件 > 新建 > 程序，然后选择将包含 EGL 文件的文件夹。
2. 选择包。
3. 指定源文件的文件名，选择 *BasicProgram* 类型并单击完成。
4. 查找程序行。
5. 在 main() 函数中，紧跟在程序行之后输入 **jas**，然后按 **Alt+/** 键以插入报告驱动程序的代码。
6. 在包含数据源连接类型和代码的窗口中，选择其中一个数据源连接类型。
7. 可修改现有代码或添加您自己的代码。如果修改代码，则插入报告驱动程序所使用的变量的特定值。这些变量包括 *reportDesignFileName*、*reportDestinationFileName*、*exportReportFile*、*alias*、*databaseName*、*userid*、*password* 和 *connectionName*。

显示报告调用信息的代码

以下代码显示报告调用信息：

```
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);
```

代码	说明
myReport Report;	这是报告库记录声明。
myReportData ReportData;	这是报告库数据记录声明。
myReport.reportDesignFile = "myReport_XML.jasper";	此语句定义要用于创建报告的报告设计。
myReport.reportDestinationFile = "myReport.jrprint";	此语句指定生成的报告输出的文件名。
myReport.reportExportFile = "myReport.pdf"	此语句指定导出的输出的文件名。
myReport.sqlStatement = "Select * From myTable";	这会提供有关报告中使用的 SQL SELECT 语句的信息。
myReport.reportData = myReportData;	这会提供有关报告数据的信息。
ReportLib.fillReport(myReport, DataSource.sqlStatement);	此语句指定报告的源信息。
ReportLib.exportReport(myReport, ExportFormat.pdf);	此语句指定报告输出格式。

示例代码片段:

```
//location where the .jprint file is stored.  
abcReport.reportDestinationFile="C:\\temp\\MasterReport.jrprint";  
  
//location for the exported report.  
abcReport.reportExportFile="C:\\temp\\MasterReport.pdf";  
  
//perform the export.  
ReportLib.exportReport(abcReport, ExportFormat.pdf);
```

相关概念

EGL 报告概述

EGL 报告创建过程概述

相关任务

创建 EGL 报告处理程序

手工创建 EGL 报告处理程序

使用报告模板

相关参考

EGL 报告处理程序

EGL 报告库

EGL 报告驱动程序函数的样本代码

为报告生成文件并运行报告

必须具有带 .jrxml 扩展名的 XML 设计文档，它会指定报告的布局和其它设计信息。

要为 EGL 项目构建并生成报告，遵循下列步骤:

1. 通过选择**项目 > 全部构建**来构建 EGL 项目。

EGL 自动从 EGL 报告处理程序生成 Java 代码并将 XML 设计文档 (.jrxml 文件) 编译成 .jasper 文件。

2. 运行具有报告调用代码的 EGL 程序。在“包资源管理器”中完成此任务的一种方法是浏览至包含该代码的 .egl 文件并右键单击该文件。然后，从弹出菜单中选择**生成**。

除了生成报告之外，EGL 使用的 JasperReports 程序还会将生成的报告自动保存在报告调用代码中的 *reportDestinationFileName* 指定的位置。

生成报告的 JasperReports 程序还会生成 .jprint 文件并存储它，这是一种中间文件格式，它将导出至最终报告格式 (.pdf、.html、.xml、.txt 或 .csv)。程序可将一个 .jprint 文件重复用于多次导出。

报告调用代码中的 *exportReport()* 函数导致 EGL 以指定格式导出报告。

相关概念

EGL 报告概述

EGL 报告创建过程概述

相关任务

创建 EGL 报告处理程序

手工创建 EGL 报告处理程序

编写用于生成报告的代码
导出报告

相关参考

EGL 报告库
EGL 报告处理程序

导出报告

可将填写的报告导出为 PDF、HTML、XML、CSV（用逗号隔开值）和明文输出。EGL 报告驱动程序代码中的 `exportReport()` 函数导致 EGL 以指定格式导出报告。

报告驱动程序代码中的 `exportReportFile` 值指定已导出文件的位置。

要指定已导出报告的格式，在 `exportReport()` 函数的调用中使用下列其中一个参数：

- `ExportFormat.html`
- `ExportFormat.pdf`
- `ExportFormat.text`
- `ExportFormat.xml`
- `ExportFormat.csv`

例如，以下代码导致 EGL 以 `.pdf` 格式导出报告：

```
reportLib.exportReport(myReport, ExportFormat.pdf);
```

要点：EGL 不会自动刷新导出的报告。如果更改报告设计或者数据更改，则必须重新填写报告并重新导出它。

相关概念

EGL 报告概述
EGL 报告创建过程概述

相关任务

创建 EGL 报告处理程序
手工创建 EGL 报告处理程序
编写用于生成报告的代码
运行报告

相关参考

EGL 报告库
EGL 报告处理程序

使用文件和数据库

SQL 支持

如下表所示，EGL 生成的代码可以访问任何目标系统上的关系数据库。

目标系统	对访问关系数据库的支持
AIX、iSeries、Linux、Windows 2000/NT/XP 和 UNIX System Services	JDBC 提供了对 DB2 UDB、Oracle 或 Informix 的访问

开发程序时，可以象使用大多数其它语言编写程序时那样编写 SQL 语句。为了方便您工作，EGL 提供了 SQL 语句模板来供您填写。

另一种方法是，在编写 EGL 语句时，可以将 SQL 记录用作 I/O 对象。以此方式使用记录意味着通过对提供给您 SQL 语句进行定制或者通过依靠缺省 SQL 语句（从而不需要编写 SQL）来访问数据库。

在任何一种情况下，您都要了解 EGL 对 SQL 支持的下列方面：

- 如果要在特定表列中测试 NULL，则必须将列值接收到 SQL 记录中，并且接收到被声明为 nullable 的记录项中。有关详细信息，请参阅后面描述的测试与设置 NULL。
- 在接下来的概述部分中（并且为了与 SQL 术语统一），将 SQL 语句中引用的每个项都称为主变量。主这个词指的是嵌入 SQL 语句的语言；在本例中，指的是 EGL 过程语言。SQL 语句中的主变量前面有一个冒号，如以下示例所示：

```
select empnum, empname
from employee
where empnum >= :myRecord.empnum
for update of empname
```

EGL 语句和 SQL

下表列示了可以用来访问关系数据库的 EGL 关键字。此表还显示了与每个关键字相对应的 SQL 语句的概述。例如，在编写 EGL **add** 语句时，将生成 SQL INSERT 语句。

在许多商业应用程序中，使用 EGL **open** 语句和各种 **get by position** 语句。这些代码帮助您声明、打开和处理游标，后者是执行下列操作的运行时实体：

- 返回结果集，这是满足搜索条件的行的列表
- 指向结果集中的特定行

可以使用 EGL **open** 语句来调用存储过程。该存储过程由 EGL 外部编写的逻辑组成的并存储在数据库管理系统中，它也返回结果集。（另外，可以使用 EGL **execute** 语句来调用存储过程。）

下列各节提供有关处理结果集的详细信息。

如果您打算显式地编写 SQL 语句，则使用 EGL **execute** 语句，并有可能使用 EGL **prepare** 语句。

关键字 / 用途	SQL 语句的概述	是否可以修改 SQL?
add 在数据库中添加一行；或者（如果使用 SQL 记录动态数组的话），根据连续数组元素的内容添加一组行。	INSERT row（如果指定动态数组，则重复地发生）。	是
close 释放未处理的行。	CLOSE cursor。	否
delete 从数据库中删除一行。	DELETE row。该行是通过下列两种方法的其中一种选择的： <ul style="list-style-type: none"> • 当您调用带有 forUpdate 选项的 get 语句时（当您希望选择带有同一键值的若干行中的第一行时，这是合适的） • 当您调用带有 forUpdate 选项的 open 语句并接着调用 get next 语句时（当您希望选择一组行并在循环中处理检索到的数据时，这是合适的） 	否
forEach 标记在循环中运行的一组语句的开始。仅当指定的结果集可用时才会发生第一次迭代，并且该迭代会持续（在大多数情况下）至处理结果集的最后一行。	EGL 将 forEach 语句转换成在循环内部运行的 SQL FETCH 语句。	否
freeSQL 释放与动态预编译 SQL 语句相关联的所有资源，关闭与该 SQL 语句相关联的任何打开游标。		否
get （也称为 get by key value ） 从数据库中读取一行；或者（如果使用 SQL 记录动态数组的话），将连续的行读入数组的连续元素。	SELECT row，但仅当设置了 singleRow 选项时才如此。否则，下列规则适用： <ul style="list-style-type: none"> • EGL 将 get 语句转换为以下内容： <ul style="list-style-type: none"> – DECLARE cursor with SELECT 或（如果设置了 forUpdate 选项的话）DECLARE cursor with SELECT FOR UPDATE。 – OPEN cursor。 – FETCH row。 • 如果未指定 forUpdate 选项，则 EGL 还关闭游标。 • 动态数组不支持 singleRow 和 forUpdate 选项；在该情况下，EGL 运行时声明并打开游标，访存一系列行，然后关闭游标。 	是

关键字 / 用途	SQL 语句的概述	是否可以修改 SQL?
<p>get absolute</p> <p>读取结果集中由 open 语句选择的用数字指定的行。</p>	EGL 将 get absolute 语句转换为 SQL FETCH 语句。	否
<p>get current</p> <p>读取结果集中由 open 语句选择的游标已经定位的行。</p>	EGL 将 get current 语句转换为 SQL FETCH 语句。	否
<p>get first</p> <p>读取结果集中由 open 语句选择的第一行。</p>	EGL 将 get first 语句转换为 SQL FETCH 语句。	否
<p>get last</p> <p>读取结果集中由 open 语句选择的最后一行。</p>	EGL 将 get last 语句转换为 SQL FETCH 语句。	否
<p>get next</p> <p>读取结果集中由 open 语句选择的下一行。</p>	EGL 将 get next 语句转换为 SQL FETCH 语句。	否
<p>get previous</p> <p>读取结果集中由 open 语句选择的上一行。</p>	EGL 将 get previous 语句转换为 SQL FETCH 语句。	否
<p>get relative</p> <p>读取结果集中由 open 语句选择的用数字指定的行。该行是根据结果集中的游标位置指定的。</p>	EGL 将 get relative 语句转换为 SQL FETCH 语句。	否
<p>execute</p> <p>允许运行 SQL 数据定义语句（例如，CREATE TABLE 类型）或数据处理语句（例如，INSERT 或 UPDATE 类型）；或者运行不以 SELECT 子句开头的预编译 SQL 语句。</p>	<p>您编写的 SQL 语句可用于数据库管理系统。</p> <p>execute 的主要用途是编写单一 SQL 语句，该 SQL 语句在生成时要定义完整的格式，如以下示例所示：</p> <pre> try execute #sql{ // no space after "#sql" delete from EMPLOYEE where department = :myRecord.department }; onException myErrorHandler(10); end </pre> <p>定义了完整格式的 SQL 语句可以在 WHERE 子句中包含主变量。</p>	是

关键字 / 用途	SQL 语句的概述	是否可以修改 SQL?
<p>open</p> <p>从关系数据库中选择一组行，以供 get next 语句以后检索。</p>	<p>EGL 将 open 语句转换为 CALL 语句（用于访问存储过程），或转换为下列语句：</p> <ul style="list-style-type: none"> • DECLARE cursor with SELECT 或 DECLARE cursor with SELECT FOR UPDATE。 • OPEN cursor。 	是
<p>prepare</p> <p>指定一个 SQL PREPARE 语句，该语句包含（可选）只有在运行时才知道的详细信息；通过运行 EGL execute 语句或者（如果 SQL 语句以 SELECT 开头的话）通过运行 EGL open 或 get 语句来运行预编译 SQL 语句。</p>	<p>EGL 将 prepare 语句转换为 SQL PREPARE 语句，后者始终是在运行时构造的。在 EGL prepare 语句的以下示例中，每个参数标记 (?) 都是由后续 execute 语句中的 USING 子句解析的：</p> <pre> myString = "insert into myTable " + "(empnum, empname) " + "value ?, ?"; try prepare myStatement from myString; onException // exit the program myErrorHandler(12); end try execute myStatement using :myRecord.empnum, :myRecord.empname; onException myErrorHandler(15); end </pre>	是
<p>replace</p> <p>将更改了的行放回数据库中。</p>	<p>UPDATE row。该行是通过下列两种方法的其中一种选择的：</p> <ul style="list-style-type: none"> • 当您调用带有 forUpdate 选项的 get 语句时（当您希望选择带有同一键值的若干行中的第一行时，这是合适的）；或者 • 当您调用带有 forUpdate 选项的 open 语句并接着调用 get next 语句时（当您希望选择一组行并在循环中处理检索到的数据时，这是合适的）。 	是

注：在任何情况下都不能通过编写单个 EGL 语句来更新多个数据库表。

结果集处理

更新一系列行的常用方法如下所示：

1. 通过运行带有 forUpdate 选项的 EGL **open** 语句来声明并打开游标；该选项导致将选择的行锁定，以便接着进行更新或删除
2. 通过运行 EGL **get next** 语句来访问行
3. 在一个循环中执行以下操作：

- a. 更改主变量中的数据（您已将数据检索到那些主变量中）
 - b. 通过运行 EGL **replace** 语句来更新行
 - c. 通过运行 EGL **get next** 语句来访问另一行
4. 通过运行 EGL 函数 **commit** 来落实更改。

打开游标的语句和对该游标的行执行操作的语句是通过结果集标识——相关的，该结果集标识在程序中的所有结果集标识、程序变量和程序参数中必须是唯一的。在打开游标的 **open** 语句中指定该标识，并在影响单个行的 **get next**、**delete** 和 **replace** 语句中以及在关闭游标的 **close** 语句中引用同一标识。有关其它详细信息，请参阅 *resultSetID*。

下列代码说明在您自己编写 SQL 时如何更新一系列行:

```

VGVar.handleHardIOErrors = 1;

    try
open selectEmp forUpdate with
#sql{          // no space after "#sql"
select empname
from EMPLOYEE
where empnum >= :myRecord.empnum
for update of empname
};

onException
myErrorHandler(8); // exits program
end

    try
get next from selectEmp into :myRecord.empname;
onException
if (sysVar.sqlcode != 100)
myErrorHandler(8); // exit the program
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); // exits program
end

    try
get next from selectEmp into :myRecord.empname;
onException
if (sysVar.sqlcode != 100)
myErrorHandler(8); // exits program
end

end

```

```

        end // end while; cursor is closed automatically
          // when the last row in the result set is read

sysLib.commit;

```

如果您希望避免上一个示例中的一些复杂性，请考虑使用 SQL 记录。通过使用 SQL 记录，可以使代码简化以及使用不随数据库管理系统的改变而变化的 I/O 错误值。下一个示例等同于上一个示例，但使用名为 emp 的 SQL 记录：

```

VGVar.handleHardIOErrors = 1;

        try
        open selectEmp forUpdate for emp;
onException
myErrorHandler(8); // exits program
end

        try
        get next emp;
onException
if (sysVar.sqlcode not noRecordFound)
myErrorHandler(8); // exit the program
end

end

while (sysVar.sqlcode not noRecordFound)
myRecord.empname = myRecord.empname + " " + "III";

        try
        replace emp;
onException
myErrorHandler(10); // exits program
end

        try
        get next emp;
        on exception
        if (sysVar.sqlcode not noRecordFound)
        myErrorHandler(8); // exits program
        end

end

        end // end while; cursor is closed automatically
          // when the last row in the result set is read

sysLib.commit;

```

下列各节描述 SQL 记录。

SQL 记录和它们的使用

SQL 记录是基于 SQL 记录部件的变量。此类型的记录允许您与关系数据库进行交互，就象是正在访问文件一样。例如，如果变量 EMP 基于引用数据库表 EMPLOYEE 的 SQL 记录部件，则可以在 EGL **add** 语句中使用 EMP：

```
add EMP;
```

在此例中，EGL 将 EMP 中的数据插入到 EMPLOYEE 中。SQL 记录还包含状态信息，因此，在 EGL 语句运行后，可以测试 SQL 记录以根据数据库访问操作所得到的 I/O 错误值来有条件地执行任务：

```
VGVar.handleHardIOErrors = 1;

    try
    add EMP;
onException
    if (EMP is unique)    // if a table row
                        // had the same key
        myErrorHandler(8);
    end
end

end
```

SQL 记录（如 EMP）允许您与关系数据库进行交互，如下所示：

- 声明 SQL 记录部件和相关的 SQL 记录
- 定义一组 EGL 语句，其中每个 EGL 语句都将 SQL 记录用作 I/O 对象
- 接受 EGL 语句的缺省行为，或进行适合于业务逻辑的 SQL 更改

声明 SQL 记录部件和相关记录

您声明 SQL 记录部件并将每个记录项与关系表或视图中的一列相关联。可以通过 EGL 编辑器的检索功能让 EGL 自动执行此关联，如下面的声明时的数据库访问中所述。

如果 SQL 记录部件不在固定记录部件中，您可以加入基本字段和其它变量。您可能特别希望加入下列几种变量：

- 其它 SQL 记录。每一种 SQL 记录表示父表与子表之间的一对一关系。
- SQL 记录数组。每一种 SQL 记录数组表示父表与子表之间的一对多关系。

只有基本类型的字段可以表示数据库列。

如果字段前有级别号，则 SQL 记录部件是固定记录部件。下列规则适用：

- 每个 SQL 记录部件中的结构都必须是平面的（无层次结构）
- 所有字段必须是基本字段，而不是类型为 BLOB、CLOB 或 STRING 的字段
- 任何记录字段都不能是结构字段数组

在声明 SQL 记录部件之后，声明基于该部件的 SQL 记录。

定义与 SQL 相关的 EGL 语句

可以定义一组 EGL 语句，每个 EGL 语句都将 SQL 记录用作语句中的 I/O 对象。对于每个语句，EGL 提供了隐式 SQL 语句，该语句并未包含在源代码中，而是由 SQL 记录与 EGL 语句的组合所隐含的。例如，对于 EGL **add** 语句，隐式 SQL INSERT 语句将给定记录项的值放入相关联的表列。如果 SQL 记录包含未指定表列的记录项，则 EGL 根据一个假定来构造隐式 SQL 语句，即假定记录项名与列名完全相同。

使用隐式 SELECT 语句： 当定义使用 SQL 记录并生成 SQL SELECT 语句或游标声明的 EGL 语句时，EGL 将提供隐式 SQL SELECT 语句。（如果有游标声明的话，该语句就嵌入在游标声明中。）例如，可以声明一个名为 EMP 并基于以下记录部件的变量：

```
Record Employee type sqlRecord
  { tableNames = [{"EMPLOYEE"}],
    keyItems = ["empnum"] }
  empnum decimal(6,0);
  empname char(40);
end
```

然后，可以编写 **get** 语句：

```
get EMP;
```

隐式 SQL SELECT 语句如下所示：

```
SELECT empnum, empname
FROM EMPLOYEE
WHERE empnum = :empnum
```

EGL 还将一个 INTO 子句放到独立的 SELECT 语句中（如果未涉及游标声明的话），或放到与游标相关联的 FETCH 语句中。INTO 子句列示主变量，那些主变量从 SELECT 语句的第一个子句中列示的列接收值：

```
INTO :empnum, :empname
```

隐式 SELECT 语句将每个列值读入相应的主变量、引用 SQL 记录中指定的表并具有一个搜索条件（WHERE 子句），该搜索条件取决于两个因素的组合：

- 对记录属性 **defaultSelectCondition** 指定的值；以及
- 两组值之间的关系（例如，等同性）：
 - 组成表键的列的名称
 - 组成记录键的主变量的值

如果将数据读入 SQL 记录动态数组（可以通过 **get** 语句做到这一点），就会发生一种特殊的情况：

- 游标被打开，数据库中连续的行被读入到数组的连续元素中，结果集被释放，游标被关闭。
- 如果未指定 SQL 语句，则搜索条件取决于记录属性 **defaultSelectCondition**，但也取决于下列各组值之间的关系（明确地说，是大于等于关系）：
 - 列的名称，这些名称是在 EGL 语句中指定项间接指定的
 - 那些项的值

属性 **defaultSelectCondition** 中指定的任何主变量都必须位于作为动态数组基础的 SQL 记录之外。

有关隐式 SELECT 语句（这种语句随着关键字的变化而变化）的详细信息，请参阅 *get* 和 *open*。

将 SQL 记录与游标配合使用： 当使用 SQL 记录时，可以通过在若干个 EGL 语句中使用同一个 SQL 记录来使游标处理语句相关，正如可以使用结果集标识做到这一点一样。但是，由结果集标识指示的任何交叉语句关系都优先于由 SQL 记录指示的关系；在某些情况下，必须指定 **resultSetID**。

另外，对于特定的 SQL 记录，只能打开一个游标。如果在已对同一个 SQL 记录打开了一个游标的情况下 EGL 语句打开另一个游标，则生成的代码会自动关闭第一个游标。

定制 SQL 语句

给定将 SQL 记录用作 I/O 对象的 EGL 语句，可以按照下列两种方式的其中一种方式继续：

- 可以接受隐式 SQL 语句。在这种情况下，对 SQL 记录部件所作的更改会影响在运行时使用的 SQL 语句。例如，如果以后指示要将另一条记录项用作 SQL 记录的键，则 EGL 会更改任何基于该 SQL 记录部件的游标声明中使用的隐式 SELECT 语句。
- 或者，您可以选择使 SQL 语句成为显式的。在这种情况下，该 SQL 语句的详细信息是与 SQL 记录部件分开的，对 SQL 记录部件所作的任何后续更改都不影响在运行时使用的 SQL 语句。

如果从源代码中除去显式 SQL 语句，则隐式 SQL 语句（如果有的话）在生成时再次可用。

使用记录中的记录的示例

要允许程序检索有关某个部门中的一系列职员的数据，可创建两个记录部件和一个函数，如下所示：

```
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
```

测试与设置 NULL

在某些情况下，EGL 在内部保留一个空指示符，以便在代码中表示变量的子集。如果接受这一缺省行为，EGL 在内部仅为具有下列特征的每个变量保留一个空指示符：

- 位于 SQL 记录中
- 是使用设置为 *yes* 的属性 **isNullable** 声明的

不要象在某些语言中那样在 SQL 语句中对空指示符编写主变量。要测试可空主变量中的 NULL，请使用 EGL **if** 语句。也可以测试被截断值的检索，但仅当空指示符可用时才能这样做。

可以通过两种方法的其中一种来使 SQL 表列为 NULL：

- 使用 EGL **set** 语句来使可空主变量为 NULL，然后将相关 SQL 记录写至数据库；或者
- 使用适当的 SQL 语法，通过从头开始编写 SQL 语句或通过定制与 EGL **add** 或 **replace** 语句相关联的 SQL 语句

有关 NULL 处理的其它详细信息，请参阅 *itemsNullable* 和 *SQL 项属性*。

声明时的数据库访问

如果（在声明时）访问数据库，并且该数据库的特征和代码在运行时访问的数据库相似时，可以获得下列益处：

- 如果所访问数据库表或视图等同于与 SQL 记录相关联的表或视图，则可以使用 EGL 部件编辑器的检索功能来创建或覆盖记录项。检索功能访问存储在数据库管理系统中的信息，以使所创建的项的数目和数据特征反映表列的数目和特征。在调用检索功能之后，可以重命名记录项、删除记录项以及对 SQL 记录进行其它更改。
- 在声明时访问具有相应结构的数据库可帮助确保 SQL 语句在运行时对于等效数据库是有效的。

检索功能将创建一些记录项，每个这样的记录项都与相关表列同名（或几乎同名）。

不能检索使用 DB2 条件 WITH CHECK OPTIONS 定义的视图。

有关使用检索功能的更多详细信息，请参阅检索 SQL 表数据。有关命名的详细信息，请参阅设置 SQL 检索首选项。

要在声明时访问数据库，可在首选项页面中指定连接信息，如设置 SQL 数据库连接首选项中所述。

相关概念

第 219 页的『动态 SQL』

第 279 页的『逻辑工作单元』

第 678 页的『resultSetID』

相关任务

第 231 页的『检索 SQL 表数据』

第 110 页的『设置 SQL 数据库连接首选项』

第 112 页的『设置 SQL 检索首选项』

相关参考

第 511 页的『add』

第 517 页的『close』

第 425 页的『数据库权限和表名』

第 230 页的『缺省数据库』

第 520 页的『delete』

第 522 页的『execute』

第 533 页的『get』

第 544 页的『get next』

第 230 页的『Informix 和 EGL』

第 359 页的『itemsNullable』

第 561 页的『open』

第 574 页的『prepare』

第 576 页的『replace』

第 680 页的『SQL 数据代码和 EGL 主变量』

第 219 页的『SQL 示例』

第 61 页的『SQL 项属性』

- 第 682 页的『SQL 记录内部结构』
- 第 683 页的『EGL 源格式的 SQL 记录部件』
- 第 217 页的『测试与设置 NULL』

动态 SQL

可以在生成时静态地指定与 EGL 语句相关联的 SQL 语句并适当地指定所有详细信息。然而，当动态 SQL 生效时，在运行时每次调用 EGL 语句时都会构建 SQL 语句。

使用动态 SQL 会降低运行时的处理速度，但是允许您改变数据库操作以响应运行时值：

- 对于数据库查询，您可能想要改变选择标准、数据聚集方式或返回行的顺序；那些详细信息是由 WHERE、HAVING、GROUP BY 和 ORDER BY 子句控制的。在这种情况下，可以使用 prepare 语句。
- 对于许多类型的操作，您可能想要一个运行时值来确定要访问的表。可以通过两种方法来实现表的动态指定：
 - 使用 prepare 语句；或者
 - 使用 SQL 记录并对 **tableNameVariables** 属性指定值，如 *EGL 源格式的 SQL 记录部件*中所述。

相关概念

第 209 页的『SQL 支持』

相关参考

第 425 页的『数据库权限和表名』

第 574 页的『prepare』

第 683 页的『EGL 源格式的 SQL 记录部件』

SQL 示例

可以按照下列任何方式来访问 SQL 数据库：

- 通过手工编写在生成时具有已知格式的 SQL 语句。
- 当 SQL 语句的格式在生成时已知时，通过使用 SQL 记录作为 EGL 语句的 I/O 对象：
 - 如果在 EGL 源代码中指定了显式 SQL 语句，则在运行时将使用该 SQL 语句；
 - 否则，在运行时将使用隐式 SQL 语句。
- 通过编写 EGL **prepare** 语句，该语句生成一个 SQL PREPARE 语句，而该 SQL PREPARE 语句又在运行时创建一个 SQL 语句。

在每一种情况下，都可以使用 SQL 记录来作为内存区并提供一种简单的方法来测试操作是否成功。本节中的示例假定在 EGL 文件中声明了一个记录部件，并且在该文件中的程序中声明了一条基于该部件的记录：

- 该 SQL 记录部件如下所示：

```
Record Employee type sqlRecord
{
    tableName = ["employee"],
    keyItems = ["empnum"],
    defaultSelectCondition =
        #sqlCondition{ // no space between #sql and the brace
            aTableColumn = 4 -- start each SQL comment
                -- with a double hyphen
        }
```

```

    }
}

empnum decimal(6,0) {isReadOnly=yes};
empname char(40);
end

```

- 该 SQL 记录如下所示:

```
emp Employee;
```

有关 SQL 记录和隐式语句的更多详细信息, 请参阅 SQL 支持。

编写 SQL 语句

要准备编写 SQL 语句, 请声明变量:

```
empnum decimal(6,0);
empname char(40);
```

向 SQL 表添加行: 要准备添加行, 请对变量赋值:

```
empnum = 1; empname = "John";
```

要添加行, 请将 EGL **execute** 语句与 SQL INSERT 语句相关联, 如下所示:

```

try
  execute
  #sql{
    insert into employee (empnum, empname)
    values (:empnum, :empname)
  };
onException
  myErrorHandler(8);
end

```

从 SQL 表读取一组行: 要准备从 SQL 表读取一组行, 请标识记录键:

```
empnum = 1;
```

要获取数据, 请编写一系列 EGL 语句:

- 要选择结果集, 请运行 EGL open 语句:

```

open selectEmp
with #sql{
  select empnum, empname
  from employee
  where empnum >= :empnum
  for update of empname
}
into empnum, empname;

```

- 要访问结果集的下一行, 请运行 EGL get next 语句:

```
get next from selectEmp;
```

如果在 open 语句中未指定 into 子句, 则需要 get next 语句中指定 into 子句; 并且, 如果在两个位置都指定了 into 子句, 则 get next 语句中的该子句具有优先权:

```

get next from selectEmp
into empnum, empname;

```

在从结果集中读取了最后一条记录之后, 游标将自动关闭。

下面的代码是较完整的示例, 它将更新一组行:

```

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); // exits program
end

try
  get next from selectEmp;
onException
  if (sqlcode != 100)
    myErrorHandler(8); // exits program
  end
end

while (sqlcode != 100)
  empname = empname + " " + "III";

  try
    execute
    #sql{
      update employee
      set empname = :empname
      where current of selectEmp
    };
  onException
    myErrorHandler(10); // exits program
  end

  try
    get next from selectEmp;
  onException
    if (sqlcode != 100)
      myErrorHandler(8); // exits program
    end
  end

end // end while; cursor is closed automatically
// when the last row in the result set is read

sysLib.commit();

```

可使用 `forEach` 语句来对结果集中的每一行执行一个语句块，而不是编写 `get next` 和 `while` 语句：

```

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); // exits program
end

try
  forEach (from selectEmp)
    empname = empname + " " + "III";

    try
      execute
        #sql{
          update employee
            set empname = :empname
          where current of selectEmp
        };
    onException
      myErrorHandler(10); // exits program
    end

    end // end forEach; cursor is closed automatically
    // when the last row in the result set is read

onException
  // the exception block related to forEach is not run if the condition
  // is "sqlcode = 100", so avoid the test "if (sqlcode != 100)"
  myErrorHandler(8); // exits program
end

sysLib.commit();

```

将 SQL 记录与隐式 SQL 语句配合使用

要开始使用 EGL SQL 记录，请声明 SQL 记录部件：

```

Record Employee type sqlRecord
{
  tableNames = [{"employee"}],
  keyItems = ["empnum"],
  defaultSelectCondition =
    #sqlCondition{
      aTableColumn = 4 -- start each SQL comment
                      -- with a double hyphen
    }
}

empnum decimal(6,0) {isReadOnly=yes};
empname char(40);
end

```

声明基于该记录部件的记录：

```
emp Employee;
```

向 SQL 表添加行： 要准备向 SQL 表添加行，请在 EGL 记录中指定值：

```
emp.empnum = 1; emp.empname = "John";
```

通过指定 EGL add 语句来将职员添加到表中：

```

try
  add emp;
onException
  myErrorHandler(8);
end

```

从 **SQL 表中读取行**: 要准备从 SQL 表中读取行, 请标识记录键:

```
emp.empnum = 1;
```

通过下列任何一种方式获取一行:

- 按照生成一系列语句 (DECLARE cursor、OPEN cursor、FETCH row 以及 CLOSE cursor (如果不存在 forUpdate 的话)) 的方式指定 EGL get 语句:

```

try
  get emp;
onException
  myErrorHandler(8);
end

```

- 按照生成单一 SELECT 语句的方式指定 EGL get 语句:

```

try
  get emp singleRow;
onException
  myErrorHandler(8);
end

```

通过下列任一方式处理多行:

- 使用 EGL open、get next 和 while 语句:

```
VGVar.handleHardIOErrors = 1;
```

```

try
  open selectEmp forUpdate for emp;
onException
  myErrorHandler(6); // exits program
end

```

```

try
  get next emp;
onException
  if (emp not noRecordFound)
    myErrorHandler(8); // exit the program
  end
end

```

```

while (emp not noRecordFound)
  myRecord.empname = myRecord.empname + " " + "III";

```

```

try
  replace emp;
onException
  myErrorHandler(10); // exits program
end

```

```

try
  get next emp;
onException
  if (emp not noRecordFound)
    myErrorHandler(8); // exits program
  end
end

```

```

        end

        end

    end // end while; cursor is closed automatically
        // when the last row in the result set is read

    sysLib.commit();

```

- 使用 EGL open 和 forEach 语句:

```

VGVar.handleHardIOErrors = 1;

try
    open selectEmp forUpdate for emp;
onException
    myErrorHandler(6); // exits program
end

try
    forEach (from selectEmp)
        myRecord.empname = myRecord.empname + " " + "III";

        try
            replace emp;
onException
            myErrorHandler(10); // exits program
        end

    end // end forEach; cursor is closed automatically
        // when the last row in the result set is read

onException

    // the exception block related to forEach is not run if the condition
    // is noRecordFound, so avoid the test "if (not noRecordFound)"
    myErrorHandler(8); // exit the program
end

sysLib.commit();

```

将 SQL 记录与显式 SQL 语句配合使用

在将 SQL 记录与显式 SQL 语句配合使用之前，请声明 SQL 记录部件。此部件与 SQL 项属性语法中的以及计算的值的使用中的上一个部件不同：

```

Record Employee type sqlRecord
{
    tableNameVariables = ["empTable"],
        // use of a table-name variable
        // means that the table is specified
        // at run time
    keyItems = ["empnum"]
}
empnum decimal(6,0) { isReadOnly = yes };
empname char(40);

// specify properties of a calculated column
aValue decimal(6,0)
{ isReadOnly = yes,
  column = "(empnum + 1) as NEWNUM" };
end

```

声明变量:

```
emp Employee;
empTable char(40);
```

向 SQL 表添加行: 要准备向 SQL 表添加行, 请在 EGL 记录中以及在表名变量中指定值:

```
emp.empnum = 1;    emp.empname = "John";
empTable = "Employee";
```

通过指定 EGL add 语句并修改 SQL 语句来将职员添加到表中:

```
// a colon does not precede a table name variable
try
  add emp
  with #sql{
    insert into empTable (empnum, empname)
    values (:empnum, :empname || ' ' || 'Smith')
  }

onException
  myErrorHandler(8);
end
```

从 SQL 表中读取行: 要准备从 SQL 表中读取行, 请标识记录键:

```
emp.empnum = 1;
```

通过下列任何一种方式获取一行:

- 按照生成一系列语句 (DECLARE cursor、OPEN cursor、FETCH row 以及 CLOSE cursor) 的方式指定 EGL get 语句:

```
try
  get emp into empname // The into clause is optional. (It
                       // cannot be in the SELECT statement.)
  with #sql{
    select empname
    from empTable
    where empnum = :empnum + 1
  }
onException
  myErrorHandler(8);
end
```

- 按照生成单一 SELECT 语句的方式指定 EGL get 语句:

```
try
  get emp singleRow // The into clause is derived
                   // from the SQL record and is based
                   // on the columns in the select clause
  with #sql{
    select empname
    from empTable
    where empnum = :empnum + 1
  }
onException
  myErrorHandler(8);
end
```

通过下列任一方式处理多行:

- 使用 EGL open、get next 和 while 语句:

```
try

  // The into clause is derived
```

```

// from the SQL record and is based
// on the columns in the select clause
open selectEmp forUpdate
  with #sql{
    select empnum, empname
    from empTable
    where empnum >= :empnum
    order by NEWNUM      -- uses the calculated value
    for update of empname
  } for emp;
onException
myErrorHandler(8);    // exits the program
end

try
  get next emp;
onException
myErrorHandler(9); // exits the program
end

while (emp not noRecordFound)
  try
    replace emp
    with #sql{
      update :empTable
      set empname = :empname || ' ' || 'III'
    } from selectEmp;

    onException
    myErrorHandler(10); // exits the program
  end

  try
    get next emp;
  onException
  myErrorHandler(9); // exits the program
  end

end // end while

// no need to say "close emp;" because emp
// is closed automatically when the last
// record is read from the result set or
// (in case of an exception) when the program ends

```

```
sysLib.commit();
```

- 使用 EGL open 和 forEach 语句:

```

try

  // The into clause is derived
  // from the SQL record and is based
  // on the columns in the select clause
  open selectEmp forUpdate
    with #sql{
      select empnum, empname
      from empTable
      where empnum >= :empnum
      order by NEWNUM      -- uses the calculated value
      for update of empname
    } for emp;

  onException
  myErrorHandler(8);    // exits the program

```

```

end

try
  forEach (from selectEmp)

    try
      replace emp
      with #sql{
        update :empTable
        set empname = :empname || ' ' || 'III'
      } from selectEmp;

    onException
      myErrorHandler(9); // exits program
    end

  end // end forEach statement, and there is
      // no need to say "close emp;" because emp
      // is closed automatically when the last
      // record is read from the result set or
      // (in case of an exception) when the program ends

  onException
    // the exception block related to forEach is not run if the condition
    // is noRecordFound, so avoid the test "if (not noRecordFound)"
    myErrorHandler(9); // exits program
  end

  sysLib.commit();

```

使用 EGL prepare 语句

在编写 EGL prepare 语句时，您可以选择使用 SQL 记录部件。请声明以下部件：

```

Record Employee type sqlRecord
{
  tableNames = [{"employee"}],
  keyItems = ["empnum"],
  defaultSelectCondition =
  #sqlCondition{
    aTableColumn = 4 -- start each SQL comment
                    -- with a double hyphen
  }
}

empnum decimal(6,0) {isReadOnly=yes};
empname char(40);
end

```

声明变量：

```

emp Employee;
empnum02 decimal(6,0);
empname02 char(40);
myString char(120);

```

向 SQL 表添加行： 在添加行之前，请对变量赋值：

```

emp.empnum = 1;   emp.empname = "John";
empnum02 = 2;
empname02 = "Jane";

```

开发 SQL 语句：

- 编写 EGL prepare 语句并引用 SQL 记录，该记录提供了可以定制的 SQL 语句:

```
prepare myPrep
  from "insert into employee (empnum, empname) " +
  "values (?, ?)" for emp;

// you can use the SQL record
// to test the result of the operation
if (emp is error)
  myErrorHandler(8);
end
```

- 另外，可以在不引用 SQL 记录的情况下编写 EGL prepare 语句:

```
myString = "insert into employee (empnum, empname) " +
  "values (?, ?)";

try
  prepare addEmployee from myString;
onException
  myErrorHandler(8);
end
```

在前面的每个例子中，EGL prepare 语句都包含将由 EGL execute 语句提供的数据的占位符。下面是 execute 语句的两个示例:

- 可以从记录（SQL 记录或别的记录）中提供值:


```
execute addEmployee using emp.empnum, emp.empname;
```
- 可以从各个项中提供值:


```
execute addEmployee using empnum02, empname02;
```

从 **SQL 表中读取行**: 要准备从 SQL 表中读取行，请标识记录键:

```
empnum02 = 2;
```

通过下列任一方式替换多行:

- 使用 EGL open、while 和 get next 语句:

```
myString = "select empnum, empname from employee " +
  "where empnum >= ? for update of empname";

try
  prepare selectEmployee from myString for emp;
onException
  myErrorHandler(8); // exits the program
end

try
  open selectEmp with selectEmployee
  using empnum02
  into emp.empnum, emp.empname;
onException
  myErrorHandler(9); // exits the program
end

try
  get next from selectEmp;
onException
  myErrorHandler(10); // exits the program
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); // exits the program
    end

    try
        get next from selectEmp;
    onException
        myErrorHandler(12); // exits the program
    end

    end // end while; close is automatic when last row is read
sysLib.commit();

```

- 使用 EGL open 和 forEach 语句:

```

myString = "select empnum, empname from employee " +
           "where empnum >= ? for update of empname";

try
    prepare selectEmployee from myString for emp;
onException
    myErrorHandler(8); // exits the program
end

try
    open selectEmp with selectEmployee
        using empnum02
        into emp.empnum, emp.empname;
onException
    myErrorHandler(9); // exits the program
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); // exits the program
        end

        end // end forEach; close is automatic when last row is read
    onException

// the exception block related to forEach is not run if the condition
// is noRecordFound, so avoid the test "if (not noRecordFound)"
myErrorHandler(12); // exits the program

```

end

```
sysLib.commit();
```

缺省数据库

缺省数据库是一个关系数据库，当与 SQL 相关的 I/O 语句在 EGL 生成的代码中运行时，并且当前没有其它数据库连接时，将访问此数据库。只要运行单元一开始，缺省数据库就可用；然而，可以动态地连接至另一个数据库以便在同一运行单元中进行后续访问，如 *VGLib.connectionService* 所述。

缺省数据库是在可选运行时属性 **vgj.jdbc.default.database** 中指定的，如果在生成时将选项 **genProperties** 设置为 GLOBAL 或 PROGRAM，则该运行时属性将从构建描述符选项 **sqlDB** 中接收生成的值。

相关概念

第 315 页的『Java 运行时属性』

第 678 页的『运行单元』

第 209 页的『SQL 支持』

相关任务

第 328 页的『设置 J2EE JDBC 连接』

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

第 240 页的『了解如何建立标准 JDBC 连接』

相关参考

第 493 页的『Java 运行时属性（详细信息）』

第 364 页的『sqlDB』

第 839 页的『connectionService()』

Informix 和 EGL

下列规则是特定于 Informix 数据库和 EGL 的：

- 由 EGL 或者由 EGL 生成的程序访问的 Informix 数据库必须启用事务。
- 如果在编写 SQL 语句并且在标识 Informix 表时使用冒号 (:)，则使用引号来将 Informix 标识与语句的余下部分隔开，如下列示例中所示：

```
INSERT INTO "myDB:myTable"  
  (myColumn) values (:myField)
```

```
INSERT INTO "myDB@myServer:myTable"  
  (myColumn) values (:myField)
```

- 如果您正在使用 EGL 的 SQL 检索功能来访问非 ANSI Informix 数据库中的数据，则应确保任何类型为 DECIMAL 的数据库列都包含小数位值。例如，不是将列定义为 DECIMAL (4)，而是将列定义为 DECIMAL (4,0)。
- 如果您打算使用 SQL 检索功能来检索作为 Informix 系统模式一部分的表中的数据，则必须设置特殊的首选项，如设置 SQL 检索首选项中所述。

相关概念

第 209 页的『SQL 支持』

相关任务

『检索 SQL 表数据』

第 112 页的『设置 SQL 检索首选项』

特定于 SQL 的任务

检索 SQL 表数据

EGL 提供了根据 SQL 表定义、视图定义或连接定义来创建 SQL 记录项的方法；有关概述，请参阅 *SQL 支持*。

执行下列操作：

1. 确保已适当地设置 SQL 首选项。有关详细信息，请参阅设置 *SQL 检索首选项*。
2. 决定执行任务的位置：
 - 在 EGL 源文件中（当开发每个 SQL 记录时）；或者
 - 在“大纲”视图中（如果已有 SQL 记录，这可能比较方便）。
3. 如果正在使用 EGL 源文件，则可以按照以下方式继续进行：
 - a. 如果没有 SQL 记录，则创建它：
 - 1) 输入 **R**，按 **Alt+/**，并在内容辅助列表中选择其中一个 SQL 表条目（通常是带有表名的 SQL 记录）。
 - 2) 输入 SQL 记录的名称；按 **Tab** 键；并输入表名或者表的逗号定界列表，或者输入视图的别名。

也可以通过输入最少的内容来创建 SQL 记录，这适用于记录名与表名相同的情况，如以下示例所示：

```
Record myTable type sqlRecord  
end
```

- b. 在记录中的任何位置单击鼠标右键。
 - c. 在上下文菜单中，单击 **SQL 记录 > 检索 SQL**。
4. 如果您正在“大纲”视图中工作，则右键单击 SQL 记录的条目，并在上下文菜单中单击**检索 SQL**。

注：不能检索使用 DB2 条件 **WITH CHECK OPTIONS** 定义的 SQL 视图。

在创建记录项之后，您可能想通过创建等同的 `dataItem` 部件来提高生产力；请参阅从 *SQL 记录部件创建 dataItem 部件的概述*。

相关概念

第 232 页的『从 SQL 记录部件创建数据项部件（概述）』

第 209 页的『SQL 支持』

相关任务

第 232 页的『从 SQL 记录部件创建数据项部件』

第 110 页的『设置 SQL 数据库连接首选项』

第 112 页的『设置 SQL 检索首选项』

相关参考

第 61 页的『SQL 项属性』

从 SQL 记录部件创建数据项部件（概述）

在 SQL 记录部件中声明结构项之后，可以在 EGL 编辑器中使用一种特殊的机制来创建等同于结构项的数据项部件。优点是可以更容易地创建非 SQL 记录（通常是基本记录）以便在运行时将数据传送至相关 SQL 记录或从相关 SQL 记录传送数据。

考虑下列结构项：

```
10 myHostVar01 CHAR(3);
10 myHostVar02 BIN(9,2);
```

您可以请求创建 `dataItem` 部件：

```
DataItem myHostVar01 CHAR(3) end

DataItem myHostVar02 BIN(9,2) end
```

另一项效果是重新编写结构项声明：

```
10 myHostVar01 myHostVar01;
10 myHostVar02 myHostVar02;
```

如此示例所示，对每个 `dataItem` 部件都指定与相关结构项相同的名称，并且每个 `dataItem` 都充当结构项的 `typedef`。每个数据项部件都还可用作其它结构项的 `typedef`。

在可以将结构项用作 `dataItem` 部件的基础之前，结构项必须具有名称，必须具有有效的基本特征，并且一定不能指向 `typedef`。

相关概念

第 209 页的『SQL 支持』

相关任务

『从 SQL 记录部件创建数据项部件』

相关参考

第 431 页的『EGL 源格式的 `DataItem` 部件』

第 683 页的『EGL 源格式的 SQL 记录部件』

从 SQL 记录部件创建数据项部件

在 SQL 记录部件中声明结构项之后，可以在 EGL 编辑器中使用一种特殊的机制来创建等同于结构项的 `dataItem` 部件。有关一般信息，请参阅从 *SQL 记录部件创建 `dataItem` 部件的概述*。

如果未显示“大纲”视图，则通过从“窗口”菜单中选择 **显示视图 > 大纲** 来打开该视图。

在“大纲”视图中，执行下列操作：

1. 对于给定的 SQL 记录部件，按住 **Ctrl** 键并同时单击您感兴趣的每个结构项。要选择给定记录中的所有结构项，请单击最上面的结构项，然后按住 **Shift** 键并同时单击最下面的结构项。
2. 右键单击所选结构项。
3. 在上下文菜单中，单击 **创建 `DataItem` 部件**。

数据项部件将被写至 EGL 源文件底部，并且每个结构项都被更改为引用等效的部件。

相关概念

第 232 页的『从 SQL 记录部件创建数据项部件（概述）』

第 209 页的『SQL 支持』

相关任务

第 231 页的『检索 SQL 表数据』

相关参考

第 683 页的『EGL 源格式的 SQL 记录部件』

从关系数据库表创建 EGL 数据部件

EGL 数据部件向导

“EGL 数据部件”向导允许您从一个或多个关系数据库表或预先存在的视图创建 SQL 记录部件以及相关的数据项部件和基于库的函数部件。

连接数据库之后，可执行下列任务：

- 指定用于从给定数据库表或视图中创建、读取、更新或删除行的 SQL 记录键字段。
- 定制用于创建、读取或更新行的显式 SQL 语句。（不能定制用于删除行的 SQL 语句）。
- 指定用于从给定数据库或视图中选择一组行的 SQL 记录键字段。
- 定制用于选择一组行的显式 SQL 语句。
- 验证并运行每个 SQL 语句

输出包括下列文件：

- 用于定义每个记录部件的 EGL 源文件
- 用于每个记录部件的 EGL 库
- 一个 EGL 源文件，该文件包含 SQL 记录部件中的结构项引用的所有数据项部件

如果选择了同一文件中的记录和库复选框，就可以减少文件数目。

相关概念

第 209 页的『SQL 支持』

相关任务

第 234 页的『对 EGL 向导创建、编辑或删除数据库连接』

『从关系数据库表创建 EGL 数据部件』

第 235 页的『在 EGL 向导中定制 SQL 语句』

从关系数据库表创建 EGL 数据部件

要在不创建单独的 Web 应用程序的情况下从关系数据库表创建 EGL 数据部件，执行下列操作：

1. 选择文件 > 新建 > 其它...。将显示用于选择向导的对话框。
2. 展开 **EGL** 并双击 **EGL 数据部件**。将显示“EGL 数据部件”对话框。
3. 输入 EGL 或 EGL Web 项目名，或者从下拉列表中选择现有项目。将在此项目中生成这些部件。
4. 从下拉列表中选择现有数据库连接或建立新的数据库连接：

- 要建立新的数据库连接，单击**添加**并与**新建数据库连接向导**交互。有关特定字段中所需的输入种类的详细信息，右键单击该字段并按 **F1** 键。
- 有关编辑或删除数据库连接的详细信息，请参阅对 *EGL 向导创建、编辑或删除数据库连接*。

与数据库建立连接后，将显示数据库表列表。

5. 如果不想接受数据项的缺省 EGL 文件名，输入新的文件名。
6. 在**选择数据**字段中，单击一个表的名称，该表所包含的列将帮助您声明数据部件。要选择多个表，在单击不同表名时按住 **Ctrl** 键。要将突出显示的名称转移至所选表的列表，单击向右箭头。
7. 对于所选的每一个表（在右边），指定要创建的 EGL 记录名或接受缺省名称。要从该列表中除去一个或多个表，使所需的条目突出显示并单击向左箭头。
8. 如果想要将库部件和 SQL 记录部件包括在同一文件中，则选择该复选框。
9. 单击**下一步**。
10. 选项卡对每个表都是可用的。在每个选项卡中，指定在读取、更新和删除各行时要使用的关键字段，然后单击向右箭头。要选择多个关键字段，在单击不同字段名时按住 **Ctrl** 键。要从右边的列表中除去关键字段，使该字段名突出显示并单击向左箭头。
11. 选择在选择一组行时要使用的选择条件字段，然后单击向右箭头。要选择多个字段，在单击不同字段名时按住 **Ctrl** 键。要从右边的列表中除去字段，使该字段名突出显示并单击向左箭头。
12. 要定制隐式 SQL 语句，请参阅在 *EGL 向导中定制 SQL 语句*。此选项对 EGL delete 语句不可用。
13. 单击**下一步**。
14. 将显示“生成 EGL 数据部件”屏幕，包括（在底部）将产生的文件列表：
 - a. 要更改将接收 EGL 部件的 EGL 项目的名称，在“目标项目”字段中输入项目名称或从相关下拉列表中选择个项目。
 - b. 要对特定类型的部件（数据或库）指定 EGL 包，在相关字段中输入包名或从相关下拉列表中选择一名称。
15. 单击**完成**。

相关概念

- 第 233 页的『EGL 数据部件向导』
- 第 171 页的『EGL 数据部件和页面向导』
- 第 209 页的『SQL 支持』

相关任务

- 第 172 页的『创建单个表 EGL Web 应用程序』
- 『对 EGL 向导创建、编辑或删除数据库连接』
- 第 235 页的『在 EGL 向导中定制 SQL 语句』

对 EGL 向导创建、编辑或删除数据库连接： 如果 EGL 向导中的第一个屏幕用于从关系数据库表中创建数据部件或用于从关系数据库表中创建 Web 应用程序，并且您处在该屏幕中，应以下列任一方式指定数据库连接：

- 从下拉列表中选择现有连接；或者
- 与**新建数据库连接向导**交互。

要使用该向导来创建连接，单击**添加**并根据需要添加信息。有关特定字段中所需的输入种类的详细信息，右键单击该字段并按 **F1** 键。

要编辑现有数据库连接，执行下列操作：

1. 选择**窗口 > 打开透视图 > 其它**。在“选择透视图”对话框中，选择**全部显示**复选框并双击**数据**。
2. 在“数据库资源管理器”视图中，右键单击数据库连接，然后选择**编辑连接**。逐步完成数据库连接向导的各个页面并在必要时更改其中的信息。要获取帮助，请按 **F1** 键。
3. 要完成编辑，单击**完成**。

要删除现有数据库连接，执行下列操作：

1. 选择**窗口 > 打开透视图 > 其它**。在“选择透视图”对话框中，选择**全部显示**复选框并双击**数据**。
2. 在“数据库资源管理器”视图中，右键单击数据库连接，然后选择**删除**。

相关概念

- 第 233 页的『EGL 数据部件向导』
- 第 171 页的『EGL 数据部件和页面向导』
- 第 209 页的『SQL 支持』

相关任务

- 第 172 页的『创建单个表 EGL Web 应用程序』
- 第 233 页的『从关系数据库表创建 EGL 数据部件』
- 第 105 页的『设置 EGL 首选项』

在 EGL 向导中定制 SQL 语句： 在使用 EGL 向导从关系表创建数据部件或从关系表创建 Web 应用程序时，可以更改与读取或更新之类的操作相关联的 SQL 语句：

1. 从“编辑操作”列表中选择一个操作，然后单击**编辑 SQL**。
2. 编辑 SQL 语句（这对除删除之外的所有操作都是可能的），然后单击**验证**。验证确保该语句具有正确的语法并且符合主变量名的规则。如果该语句包含错误，将显示一条消息。更正错误并再次进行验证。

还原为上一版本将语句更改回上一有效修改版本。关闭对话框之后，先前的版本就变得不可用。

3. 单击**执行**，然后再次单击**执行**。
4. 如果 SQL 语句的主变量需要值，将显示“指定变量值”对话框。双击**值**字段以输入主变量的值，然后按 **Enter** 键。对所有主变量输入值之后，单击**完成**。

注：对于被定义为字符类型的主变量，必须用单引号将该值括起来。

5. 执行完 SQL 语句后，单击**关闭**。
6. 编辑完 SQL 语句后，单击**确定**。

相关概念

- 第 233 页的『EGL 数据部件向导』
- 第 171 页的『EGL 数据部件和页面向导』
- 第 209 页的『SQL 支持』

相关任务

- 第 172 页的『创建单个表 EGL Web 应用程序』
- 第 233 页的『从关系数据库表创建 EGL 数据部件』
- 第 234 页的『对 EGL 向导创建、编辑或删除数据库连接』
- 第 105 页的『设置 EGL 首选项』

查看 SQL 记录的 SQL SELECT 语句

EGL 为给定的 SQL 记录部件提供隐式 SQL SELECT 语句。要查看隐式 SQL SELECT 语句，执行下列操作：

1. 打开包含 SQL 记录部件的 EGL 文件。如果尚未打开该文件，则在“项目资源管理器”中右键单击 EGL 文件，然后选择打开方式 > **EGL 编辑器**。
2. 在 SQL 记录部件内单击，然后单击鼠标右键。将显示上下文菜单。
3. 选择 **SQL 记录** > **查看缺省选择**。
4. 要对数据库验证 SQL SELECT 语句，请单击**验证**。

注：在使用验证功能之前，DB2 UDB 用户必须设置 DEFERREDPREPARE 选项。可以在 CLP (DB2 命令行处理器) 中使用 **db2 update cli cfg for section COMMON using DEFERREDPREPARE 0** 命令来以交互方式设置此选项。此命令将把关键字放在 COMMON 节下面。执行命令 **db2 get cli cfg for section common** 来验证是否已检取关键字。

相关概念

第 209 页的『SQL 支持』

相关任务

『验证 SQL 记录的 SQL SELECT 语句』

相关参考

第 683 页的『EGL 源格式的 SQL 记录部件』

验证 SQL 记录的 SQL SELECT 语句

EGL 为给定的 SQL 记录部件提供隐式 SQL SELECT 语句。要对数据库验证隐式 SQL SELECT 语句，执行下列操作：

1. 打开包含 SQL 记录部件的 EGL 文件。如果尚未打开该文件，则在“项目资源管理器”中右键单击 EGL 文件，然后选择打开方式 > **EGL 编辑器**。
2. 在 SQL 记录部件内单击，然后单击鼠标右键。将显示上下文菜单。
3. 选择 **SQL 记录** > **验证缺省选择**。

注：在使用验证功能之前，DB2 UDB 用户必须设置 DEFERREDPREPARE 选项。可以在 CLP (DB2 命令行处理器) 中使用 **db2 update cli cfg for section COMMON using DEFERREDPREPARE 0** 命令来以交互方式设置此选项。此命令将把关键字放在 COMMON 节下面。执行命令 **db2 get cli cfg for section common** 来验证是否已检取关键字。

相关概念

第 209 页的『SQL 支持』

相关任务

第 236 页的『查看 SQL 记录的 SQL SELECT 语句』

相关参考

第 683 页的『EGL 源格式的 SQL 记录部件』

构造 EGL prepare 语句

在函数中，可以构造下列几种基于 SQL 记录部件的 EGL 语句：

- EGL **prepare** 语句；以及
- 相关的 EGL **execute**、**open** 或 **get** 语句。

执行下列操作：

1. 使用 EGL 编辑器打开 EGL 文件。该文件必须包含函数以及编码的 SQL 语句。如果尚未打开文件，则在工作台的“项目资源管理器”中右键单击 EGL 文件，然后选择打开方式 > **EGL 编辑器**。
2. 单击函数中要放置 EGL **prepare** 语句的位置，然后单击鼠标右键。将显示上下文菜单。
3. 选择添加 **SQL Prepare 语句**。
4. 输入一个名称以标识 EGL **prepare** 语句。有关规则，请参阅命名约定。
5. 如果已定义了 SQL 记录变量，则从下拉列表中选择它。将显示相应的 SQL 记录部件名。如果未定义任何 SQL 记录变量，则可以在 SQL 记录变量名字段中输入一个名称，然后使用浏览按钮来选择 SQL 记录部件名。最终必须在 EGL 源代码中定义具有该名称的 SQL 记录变量。
6. 从下拉列表中选择执行语句类型。
7. 如果执行语句具有 open 类型，则输入结果集标识。
8. 单击**确定**。将在函数中构造 EGL 语句。

相关概念

第 209 页的『SQL 支持』

相关任务

第 236 页的『验证 SQL 记录的 SQL SELECT 语句』

第 236 页的『查看 SQL 记录的 SQL SELECT 语句』

相关参考

第 612 页的『命名约定』

第 683 页的『EGL 源格式的 SQL 记录部件』

从隐式 SQL 语句构造显式 SQL 语句

EGL 为每个与 SQL 相关的 EGL 输入/输出 (I/O) 语句提供了隐式 SQL 语句。要根据隐式 SQL 语句构造显式 SQL 语句，执行下列操作：

1. 打开包含 EGL I/O 语句的 EGL 文件。如果尚未打开该文件，则在“项目资源管理器”中右键单击 EGL 文件，然后选择打开方式 > **EGL 编辑器**。
2. 单击 EGL I/O 语句，然后单击鼠标右键。将显示上下文菜单。

3. 要构造不带 INTO 子句的显式 SQL 语句，请选择 **SQL 语句 > 添加**。要构造带有 INTO 子句的显式 SQL 语句，请选择 **SQL 语句 > 带有 Into 的添加**。隐式 SQL 语句被追加至 EGL I/O 语句，这使后者成为显式 SQL 语句。

注： INTO 子句仅对 **open**、**get** 和 **get next** 语句有效。

相关概念

第 209 页的『SQL 支持』

相关任务

第 239 页的『从与 SQL 相关的 EGL 语句中除去 SQL 语句』

第 239 页的『重置显式 SQL 语句』

『验证隐式或显式 SQL 语句』

『查看与 SQL 相关的 EGL 语句的隐式 SQL』

查看与 SQL 相关的 EGL 语句的隐式 SQL

EGL 为每个与 SQL 相关的 EGL 输入/输出 (I/O) 语句提供了隐式 SQL 语句。要查看 EGL I/O 语句的隐式 SQL，执行下列操作：

1. 打开包含 EGL I/O 语句的 EGL 文件。如果尚未打开该文件，则在“项目资源管理器”中右键单击 EGL 文件，然后选择**打开方式 > EGL 编辑器**。
2. 单击 EGL I/O 语句，然后单击鼠标右键。将显示上下文菜单。
3. 选择 **SQL 语句 > 查看**。

相关概念

第 209 页的『SQL 支持』

相关任务

第 237 页的『从隐式 SQL 语句构造显式 SQL 语句』

第 239 页的『从与 SQL 相关的 EGL 语句中除去 SQL 语句』

第 239 页的『重置显式 SQL 语句』

『验证隐式或显式 SQL 语句』

验证隐式或显式 SQL 语句

要对数据库验证隐式或显式 SQL 语句，执行下列操作：

1. 打开包含与 SQL 相关的 EGL 语句或显式 SQL 语句的 EGL 文件。如果尚未打开该文件，则在“项目资源管理器”中右键单击 EGL 文件，然后选择**打开方式 > EGL 编辑器**。
2. 单击 EGL 语句或 SQL 语句，然后单击鼠标右键。将显示上下文菜单。
3. 选择 **SQL 语句 > 验证**。

注： 在使用验证功能之前，DB2 UDB 用户必须设置 DEFERREDPREPARE 选项。可以在 CLP (DB2 命令行处理器) 中使用 **db2 update cli cfg for section COMMON using DEFERREDPREPARE 0** 命令来以交互方式设置此选项。此命令将把关键字放在 COMMON 节下面。执行命令 **db2 get cli cfg for section common** 来验证是否已检取关键字。

相关概念

第 209 页的『SQL 支持』

相关任务

第 237 页的『从隐式 SQL 语句构造显式 SQL 语句』

『从与 SQL 相关的 EGL 语句中除去 SQL 语句』

『重置显式 SQL 语句』

第 238 页的『查看与 SQL 相关的 EGL 语句的隐式 SQL』

重置显式 SQL 语句

EGL 为每个与 SQL 相关的 EGL 输入/输出 (I/O) 语句提供了隐式 SQL 语句。隐式 SQL 语句可以被追加至 EGL I/O 语句，这使后者成为显式 SQL 语句。如果更改了显式 SQL 语句，则执行如下操作以返回基于隐式 SQL 的 SQL 语句：

1. 打开包含显式 SQL 语句的 EGL 文件。如果尚未打开该文件，则在“项目资源管理器”中右键单击 EGL 文件，然后选择打开方式 > **EGL 编辑器**。
2. 单击显式 SQL 语句，然后单击鼠标右键。将显示上下文菜单。
3. 选择 **SQL 语句 > 重置**。

相关概念

第 209 页的『SQL 支持』

相关任务

第 237 页的『从隐式 SQL 语句构造显式 SQL 语句』

『从与 SQL 相关的 EGL 语句中除去 SQL 语句』

第 238 页的『验证隐式或显式 SQL 语句』

第 238 页的『查看与 SQL 相关的 EGL 语句的隐式 SQL』

从与 SQL 相关的 EGL 语句中除去 SQL 语句

EGL 为每个与 SQL 相关的 EGL 输入/输出 (I/O) 语句提供了隐式 SQL 语句。隐式 SQL 语句可以被追加至 EGL I/O 语句，这使后者成为显式 SQL 语句（请参阅根据隐式 SQL 语句构造显式 SQL 语句）。要除去追加的 SQL 语句，执行下列操作：

1. 打开包含显式 SQL 语句的 EGL 文件。如果尚未打开该文件，则在“项目资源管理器”中右键单击 EGL 文件，然后选择打开方式 > **EGL 编辑器**。
2. 单击显式 SQL 语句，然后单击鼠标右键。将显示上下文菜单。
3. 选择 **SQL 语句 > 除去**。EGL I/O 语句将保留下来。

相关概念

第 209 页的『SQL 支持』

相关任务

第 237 页的『从隐式 SQL 语句构造显式 SQL 语句』

『重置显式 SQL 语句』

第 238 页的『验证隐式或显式 SQL 语句』

第 238 页的『查看与 SQL 相关的 EGL 语句的隐式 SQL』

解析引用以显示隐式 SQL 语句

考虑指定以下 EGL 语句时发生的情况：

```
open myRecord;
```

当 EGL 编辑器尝试创建缺省 SQL 语句时，该编辑器尝试查找名为 `myRecord` 的变量并标识该变量所基于的 SQL 记录部件。如果该变量在开发时不可用，或者未声明该变量，则编辑器尝试使用名为 `myRecord` 的 SQL 记录部件来作为缺省 SQL 语句的基础。编辑器假定您打算创建与 SQL 记录部件同名的变量。

如果您希望将与 SQL 相关的函数存储在不包含 `myRecord` 变量的文件中，则可以执行如下操作：

1. 在程序部件中，声明该全局变量
2. 将该函数作为程序部件中的嵌套函数创建
3. 创建缺省的 SQL 语句并对其进行适当的修改；然后，保存文件
4. 将该函数移动到其它文件中

在从程序部件中移动函数之后，就无法在开发时解析记录名，并且编辑器不能显示任何基于该记录的缺省 SQL 语句。

相关概念

第 209 页的『SQL 支持』

了解如何建立标准 JDBC 连接

在运行时，如果正在调试生成的 Java 程序并且如果程序属性文件包括必需的值，则会创建标准 JDBC 连接。有关程序属性的含义的详细信息（包括有关如何派生值的详细信息），请参阅 *Java 运行时属性*（[详细信息](#)）。

JDBC 连接基于下列类型的信息：

连接 URL

如果代码在调用系统函数 `sysLib.connect` 或 `VGLib.connectionService` 之前尝试访问数据库，则连接 URL 是属性 `vgj.jdbc.default.database` 的值。

如果代码尝试访问数据库来作为对调用系统函数 `sysLib.connect` 或 `VGLib.connectionService` 的响应，则连接 URL 是属性 `vgj.jdbc.databaseSN` 的值。

有关连接 URL 的格式的详细信息，请参阅 [sqlValidationConnectionURL](#)。

用户标识

如果代码在调用系统函数 `sysLib.connect` 或 `VGLib.connectionService` 之前尝试访问数据库，则用户标识是属性 `vgj.jdbc.default.userid` 的值。

如果代码尝试访问数据库来作为对那些系统函数的其中一个系统函数的响应，则用户标识是在调用中指定的值。

密码

如果代码在调用系统函数 `sysLib.connect` 或 `VGLib.connectionService` 之前尝试访问数据库，则密码是属性 `vgj.jdbc.default.password` 的值。

如果代码尝试访问数据库来作为对那些系统函数的其中一个系统函数的响应，则密码是在调用中指定的值。可以使用系统函数来避免在程序属性文件中暴露密码。

JDBC 驱动程序类

JDBC 驱动程序类是属性 `vgj.jdbc.drivers` 的值。

相关概念

第 317 页的『程序属性文件』

相关任务

第 328 页的『设置 J2EE JDBC 连接』

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

相关参考

第 820 页的『connect()』

第 839 页的『connectionService()』

第 358 页的『genProperties』

第 493 页的『Java 运行时属性（详细信息）』

第 510 页的『EGL 中的 JDBC 驱动程序需求』

第 364 页的『sqlDB』

第 364 页的『sqlID』

第 366 页的『sqlPassword』

第 366 页的『sqlValidationConnectionURL』

第 365 页的『sqlJDBCClass』

VSAM 支持

EGL 生成的 Java 代码的 VSAM 支持如下所示:

- 基于 AIX 的代码可以访问本地 VSAM 文件
- 下列代码可以访问 z/OS 上的远程 VSAM 文件:
 - EGL 生成的 Java 代码（在 Windows 2000/NT/XP 上运行）
 - EGL 调试器（在 Windows 2000/NT/XP 上运行）

访问先决条件

要进行访问，首先要在放置 VSAM 文件的系统上定义该文件。从 Windows 2000/NT/XP 进行的远程访问（无论是为 EGL 调试器进行的远程访问还是在运行时进行的远程访问）还要求在工作站上安装分布式文件管理器（DFM），如下所示:

1. 在 EGL 安装目录中定位以下文件:

```
workbench\bin\VSAMWIN.zip
```

2. 将该文件解压缩到一个新目录中，并遵循 INSTALL.README 文件中的指示信息。

系统名称

要访问本地 VSAM 文件，在资源关联部件中指定系统名称，并使用适用于操作系统的命名约定。要从 EGL 调试器中或从 EGL 生成的 Java 代码中访问远程 VSAM 文件，则按照以下方式指定系统名称:

```
\machineName\qualifier.fileName
```

machineName

SNA 配置中指定的 SNA LU 别名

qualifier.fileName

VSAM 数据集名称，包括限定符

命名约定类似于通用命名约定（UNC）格式。有关 UNC 格式的详细信息，参阅分布式文件管理器用户指南，它位于 EGL 安装目录中的以下文件中:

```
workbench\bin\VSAMWIN.zip
```

MQSeries 支持

EGL 支持访问任何目标平台上的 MQSeries 消息队列。可以按下列任何方式来提供这样的访问:

- 对 MQ 记录使用与 MQSeries 相关的 EGL 关键字, 如 **add** 和 **get next**; 在这种情况下, EGL 隐藏 MQSeries 的详细信息, 以便您可以将注意力放在代码所要解决的业务问题上
- 调用直接调用 MQSeries 命令的 EGL 函数, 在这种情况下, 可以使用某些不受 EGL 关键字支持的命令

可以在给定程序中混合使用这两种方法。但是, 在大多数情况下, 只需单独使用其中一种方法。

不论采用哪种方法, 您都可以通过定制选项记录来控制各种运行时条件, 这些记录是 EGL 运行时服务在调用时传递给 MQSeries 的全局基本记录。当您声明选项记录作为程序变量时, 可以将 EGL 安装的选项记录部件用作 `typedef`; 也可以将已安装的部件复制到您自己的 EGL 文件中, 定制该部件, 并将定制的部件用作 `typedef`。

您的方法将确定 EGL 运行时服务如何使选项记录可用于 MQSeries:

- 如果正在使用 EGL **add** 和 **get next** 语句, 则在指定 MQ 记录的属性时标识选项记录。如果未标识特定的选项记录, 则 EGL 使用缺省值。
- 如果正在调用直接调用 MQSeries 的 EGL 函数, 则调用函数时将选项记录用作自变量。在这种情况下, 缺省值不可用。

有关选项记录以及有关缺省情况下传递给 MQSeries 的值的详细信息, 请参阅 *MQ 记录的选项记录*。有关 MQSeries 本身的详细信息, 参阅下列文档:

- *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)

连接

在第一次连接至队列管理器 (称为连接队列管理器) 时, 应调用以下列表中的语句:

- 访问消息队列的 EGL **add** 或 **get next** 语句
- 调用 EGL 函数 MQCONN 或 MQCONNX

每次只能访问一个连接队列管理器; 但是, 可以访问多个受连接队列管理器控制的队列。如果要直接连接至除当前连接队列管理器之外的队列管理器, 则必须通过调用 MQDISC 来断开与第一个队列管理器的连接, 然后通过调用 **add**、**get next**、MQCONN 或 MQCONNX 来连接至第二个队列管理器。

也可以访问远程队列管理器 (连接队列管理器可与之进行交互的队列管理器) 控制之下的队列。仅当将 MQSeries 本身配置为允许两个队列管理器之间的访问时, 才有可能进行该访问。

当调用 MQDISC 时或当代码结束时, 对连接队列管理器的访问就会终止。

将消息包括在事务中

可以将队列访问语句嵌入在工作单元中，以便在单个处理点落实或回滚对队列所作的
所有更改。如果语句在工作单元中，则下列情况成立：

- 仅当进行落实时，EGL **get next** 语句（或 EGL MQGET 调用）才会除去消息
- 仅当进行落实时，才能在工作单元之外看见由 EGL **add** 语句（或 EGL MQPUT 调用）放置在队列上的消息

当队列访问语句不在工作单元中时，将立即落实对消息队列所作的每项更改。

如果属性 **includeMsgInTransaction** 对 MQ 记录有效，则与 MQSeries 相关的 EGL **add** 或 **get next** 语句将嵌入在工作单元中。生成的代码包含下列选项：

- 对于 MQGET，包含 MQGMO_SYNCPOINT
- 对于 MQPUT，包含 MQPMO_SYNCPOINT

如果未对 MQ 记录指定属性 **includeMsgInTransaction**，则队列访问语句将在工作单元之外运行。生成的代码包含下列选项：

- 对于 MQGET，包含 MQGMO_NO_SYNCPOINT
- 对于 MQPUT，包含 MQPMO_NO_SYNCPOINT

当代码结束工作单元时，EGL 将落实或回滚您的程序正在访问的所有可恢复的资源，包括数据库、消息队列和可恢复的文件。不管您是使用系统函数（**sysLib.commit** 和 **sysLib.rollback**）还是使用对 MQSeries 的 EGL 调用（MQCMIT 和 MQBACK），都会产生这种结果；在任一种情况下都会调用适当的 EGL 系统函数。

如果 EGL 程序由于 EGL 运行时服务检测到的错误而过早终止，则会进行回滚。

定制

如果要定制与 MQSeries 之间进行的交互而不是依赖 **add** 和 **get next** 语句的缺省处理，则您需要查看本节中的信息。

EGL dataTable 部件

提供了一组 EGL dataTable 部件来帮助您与 MQSeries 进行交互。每个部件都允许 EGL 提供的函数在运行时从基于内存的列表中检索值。下一节讨论有关如何部署数据表的详细信息。

使定制成为可能

要使定制成为可能，您必须将各种已安装的 EGL 文件引入到项目中，而不以任何方式更改它们。这些文件如下所示：

records.egl

包含可以用作在程序中使用的选项记录的 typedef 的基本记录部件；还包含由那些记录使用并且使您能够灵活开发自己的记录部件的结构部件

functions.egl

包含两组函数：

- MQSeries 命令函数，它们直接访问 MQSeries
- 初始化函数，它们允许您在程序中使用的选项记录中设置初始值

mqrcode.egl、mqrc.egl 和 mqvalue.egl

包含一组由命令和初始化函数使用的 EGL dataTable 部件

您的任务如下所示:

1. 通过使用将文件导入工作台的过程, 将这些文件置于 EGL 项目中。这些文件位于以下目录中:

```
installationDir\egl\eclipse\plugins\  
com.ibm.etools.egl.generators_version\MqReusableParts
```

installationDir

产品安装目录, 如 C:\Program Files\IBM\MSPD\6.0。如果在安装您要使用的产品之前安装了 Rational Developer 产品并将它保留下来, 则可能需要指定在之前安装中使用的目录。

version

插件的最新版本; 例如, 6.0.0

2. 为了使程序可以更方便地使用这些部件, 请在包含程序的文件中编写一个或多个 *EGL import* 语句。如果要导入的文件位于除正在开发代码的项目之外的项目中, 则应确保您的项目引用那个项目。

有关详细信息, 请参阅导入。

3. 在您的程序中, 声明全局变量:

- 声明 MQRC、MQRCODE 和 MQVALUE, 每个这样的变量都必须将与该变量同名的 dataTable 部件用作 typedef。
- 对于要传递给 MQSeries 的每个选项记录, 声明将选项记录部件用作 typedef 的基本记录。有关每个部件的详细信息, 请参阅 *MQ 记录的选项记录*。

4. 在函数中, 初始化要传递给 MQSeries 的选项记录。可以通过为给定选项记录调用导入的 EGL 初始化函数来方便地完成此任务。每个函数的名称就是用作记录 typedef 的部件的名称再后跟 *_INIT*。一个示例是 *MQGMO_INIT*。

5. 设置选项记录中的值。在许多情况下, 可以通过指定表示常量的 EGL 符号来设置值, 每个这样的值都基于 MQSeries 文档中描述的符号。可以通过对各个 EGL 符号进行求和来指定多个 EGL 符号, 如下示例所示:

```
MQGMO.GETOPTIONS = MQGMO_LOCK  
                  + MQGMO_ACCEPT_TRUNCATED_MSG  
                  + MQGMO_BROWSE_FIRST
```

6. 第一次生成使用 MQSeries 支持的定制功能的特定程序时, 还将生成由该程序使用的数据表。要生成在程序中使用的所有数据表, 允许构建描述符选项 **genTables** 缺省为 YES。有关其它详细信息, 请参阅 *DataTable* 部件。

与 MQSeries 相关的 EGL 关键字

当使用与 MQSeries 相关的 EGL 关键字 (例如 *add* 和 *scan*) 时, 需要为您想要访问的每个消息队列定义 MQ 记录。记录布局就是消息的格式。

下表列示了关键字。

关键字	用途
add	<p>将 MQ 记录的内容置于指定队列的末尾。</p> <p>EGL add 语句可以调用最多三个 MQSeries 命令:</p> <ul style="list-style-type: none"> • MQCONN 将生成的代码与队列管理器连接, 当没有连接处于活动状态时, 将调用此命令。 • MQOPEN 与队列建立连接, 当有连接处于活动状态, 但是未打开队列时, 将调用此命令。 • MQPUT 将记录放到队列中, 除非在先前的 MQSeries 调用中发生了错误, 否则总是会调用此命令。 <p>在添加 MQ 记录之后, 在从同一队列中读取 MQ 记录之前必须先关闭消息队列。</p>
close	<p>放弃对与 MQ 记录相关联的消息队列的访问。</p> <p>EGL close 语句调用 MQSeries MQCLOSE 命令, 当程序结束时, 也会自动调用该命令。</p> <p>如果另一个程序需要访问该消息队列, 则在执行 add 或 scan 之后应该关闭该队列。如果程序运行了很长时间并且不再需要进行访问时, 就尤其应当进行关闭。</p>
scan	<p>将队列中的第一条消息读取到消息队列记录中, (在缺省情况下) 并从队列中除去该消息。</p> <p>EGL scan 语句可以调用最多三个 MQSeries 命令:</p> <ul style="list-style-type: none"> • MQCONN 将生成的代码与队列管理器连接, 当没有连接处于活动状态时, 将调用此命令。 • MQOPEN 与队列建立连接, 当有连接处于活动状态, 但是未打开队列时, 将调用此命令。 • MQGET 从队列中除去记录, 除非在先前的 MQSeries 调用中发生了错误, 否则总是会调用此命令。 <p>在读取 MQ 记录之后, 在将 MQ 记录添加到同一队列之前必须关闭该队列。</p>

管理器和队列规范

当使用与 MQSeries 相关的 EGL 关键字时, 在下列情况下需要标识队列:

- 在声明时, 需要指定逻辑队列名, 并且这是通过设置 MQ 记录部件的 **queueName** 属性完成的。该逻辑队列名充当在运行时访问的队列名的缺省值; 但在大多数情况下, 该名称只有在作为将 MQ 记录与物理队列相关联的方法时才有意义。逻辑队列名的长度不能超过 8 个字符。
- 在生成时, 使用 **buildDescriptor** 部件来控制生成过程, 而该部件又可以引用资源关联部件。资源关联部件将队列名与物理队列的名称相关联。
- 在运行时, 代码可以更改特定于记录的变量 **record.resourceAssociation** 中的值以覆盖您在声明时或生成时指定的任何队列名。

物理队列的名称具有以下格式:

queueManagerName:physicalQueueName

queueManagerName

队列管理器的名称；如果省略此名称，则也将省略冒号

physicalQueueName

指定的队列管理器所知的物理队列名称

当第一次对消息队列记录发出 `add` 或 `scan` 语句时，必须指定连接队列管理器，可以将它作为缺省值，也可以不作为缺省值。在最简单的情况下，完全不指定连接队列管理器，而是依赖于 MQSeries 配置中的缺省值。

特定于记录的变量 **`record.resourceAssociation`** 始终至少包含给定 MQ 记录的消息队列的名称。

远程消息队列

如果要访问受远程队列管理器控制的队列，则必须执行下列操作：

- 发出 `EGL close` 语句以放弃对现在正在使用的队列的访问
- 设置特定于记录的变量 **`record.resourceAssociation`** 以确保以后可以访问远程队列

根据在 MQSeries 中建立队列管理器关系的方式的不同，使用下列两种方法的其中之一来设置 **`record.resourceAssociation`**：

- 如果连接队列管理器具有远程队列的本地定义，则按如下方式设置 **`record.resourceAssociation`**：
 - 对连接队列管理器接受同一个值（通过指定连接队列管理器的名称，或者通过不指定任何名称；在后一种情况下，省略冒号）。
 - 指定远程队列的本地定义的名称。

然后使用 `add` 或 `scan` 语句发出 `MQOPEN` 以建立对远程队列的访问。

- 另外，利用远程队列管理器的名称和远程队列的名称来设置 **`record.resourceAssociation`**。在这种情况下，不会更改连接队列管理器。接着，使用 `add` 或 `scan` 语句发出 `MQOPEN` 并使用已存在的连接。

相关概念

『直接 MQSeries 调用』

第 242 页的『MQSeries 支持』

相关参考

第 606 页的『MQ 记录属性』

第 606 页的『MQ 记录的选项记录』

直接 MQSeries 调用

可以使用介于代码与 MQSeries 之间的一组已安装 EGL 函数，如 *MQSeries* 支持中所述。

下表列示可用的函数并标识必需的自变量。例如，`MQBACK` (`MQSTATE`) 指示在调用 `MQBACK` 时需要传递基于 `MQSTATE` 记录部件的自变量。稍后会描述记录部件。

与 MQSeries 相关的 EGL 函数调用	作用
MQBACK (MQSTATE)	调用系统函数 sysLib.rollback 以回滚逻辑工作单元。回滚会影响程序正在访问的所有可恢复的资源，包括数据库、消息队列和可恢复的文件。
MQBEGIN (MQSTATE, MQBO)	开始逻辑工作单元。
MQCHECK_COMPLETION (MQSTATE)	设置基于 MQSTATE 的记录的 mqdescription 字段。设置是根据最新返回的原因码进行的。将从 EGL 函数 MQBEGIN、MQCLOSE、MQCONN、MQCONNX、MQDISC、MQGET、MQINQ、MQOPEN、MQPUT、MQPUT1 和 MQSET 自动调用函数 MQCHECK_COMPLETION。
MQCLOSE (MQSTATE)	关闭 MQSTATE.hobj 引用的消息队列。
MQCMIT (MQSTATE)	调用系统函数 sysLib.commit 以落实逻辑工作单元。落实将影响程序正访问的所有可恢复的资源，包括数据库、消息队列和可恢复的文件。
MQCONN (MQSTATE, qManagerName)	连接至队列管理器，它由 qManagerName（一个最长可达 48 个字符的字符串）标识。MQSeries 设置连接句柄（MQSTATE.hconn）以供后续调用使用。 注： 可以将代码以每次连接至一个队列管理器的方式来连接至队列管理器。
MQCONNX(MQSTATE, qManagerName, MQCNO)	使用用于控制调用工作方式的选项来连接至队列管理器。队列管理器由 qManagerName（一个最长可达 48 个字符的字符串）标识。MQSeries 设置连接句柄（MQSTATE.hconn）以供后续调用使用。
MQDISC (MQSTATE)	与队列管理器断开连接。
MQGET(MQSTATE, MQMD, MQGMO, BUFFER)	从队列中读取然后除去消息。缓冲区不能超过 32767 个字节，但是如果您正在使用 EGL get next 语句，则该限制不适用。
MQINQ(MQSTATE, MQATTRIBUTES)	请求队列属性。
MQNOOP()	只由 EGL 使用。
MQOPEN(MQSTATE, MQOD)	打开消息队列。MQSeries 设置队列句柄（MQSTATE.hobj）以供后续调用使用。
MQPUT(MQSTATE, MQMD, MQPMO, BUFFER)	将消息添加至队列。缓冲区不能超过 32767 个字节，但是如果您正在使用 EGL add 语句，则该限制不适用。
MQPUT1(MQSTATE, MQOD, MQMD, MQPMO, BUFFER)	打开队列，写入一条消息，然后关闭队列。
MQSET(MQSTATE, MQATTRIBUTES)	设置队列的属性。

下表列示在调用与 MQSeries 相关的 EGL 函数时用作自变量的选项记录。还列示了应该对给定自变量调用的初始化函数。

第一步是初始化基于 MQSTATE 记录部件的自变量。在以下示例（如后面的表中所示）中，假定自变量名与记录部件的名称相同：

```
MQSTATE_INIT(MQSTATE);
```

自变量 (记录部件名)	初始化函数	描述
MQATTRIBUTES	无	属性和属性选择器的数组以及在命令 MQINQ 或 MQSET 中使用的其它信息
MQBO	MQBO_INIT (MQBO)	开始选项
MQCNO	MQCNO_INIT (MQCNO)	连接选项
MQGMO	MQGMO_INIT (MQGMO)	获取消息选项
MQIIH	MQIIH_INIT (MQIIH)	IMS™ 信息头; 描述在发送至 IMS 的 MQSeries 消息的开头所需的信息 (MQSeries 文档指示在 Windows 2000/NT/XP 上不支持使用此信息头)
MQINTATTRS	无	整数属性的数组, 用于命令 MQINQ 或 MQSET 中
MQMD	MQMD_INIT (MQMD, MQSTATE)	消息描述符 (MQSeries V2)
MQMDE	MQMDE_INIT (MQMDE, MQSTATE)	消息描述符扩展 (仅使用 MQSeries V2 中的字段)
MQOD	MQOD_INIT (MQOD)	对象描述符
MQOO	MQOO_INIT (MQOO)	打开选项
MQPMO	MQPMO_INIT (MQPMO)	放置消息选项
MQSELECTORS	无	属性选择器的数组, 仅当您希望在不使用 EGL 函数的情况下访问 MQSeries 时才使用
MQSTATE	MQSTATE_INIT (MQSTATE)	在对 MQSeries 的一个或多个调用中使用的自变量的集合; 例如, 当使用 EGL 函数 MQCONN 或 MQCONNEX 进行连接时, MQSeries 设置连接句柄 (MQSTATE.hconn) 以供后续调用使用
MQXQH	MQXQH_INIT (MQXQH, MQSTATE)	传输队列头

注: 每个记录部件都只包含一个结构项, 而结构项将结构部件用作 typedef。此设置提供了最大程度的灵活性。您可以创建自己的记录部件, 而每个记录部件都由一系列结构部件组成。

每个结构部件的名称都是记录部件的名称后跟 _S; 例如, 记录部件 MQGMO 使用名为 MQGMO_S 的结构部件。

相关概念

第 244 页的『与 MQSeries 相关的 EGL 关键字』

第 242 页的『MQSeries 支持』

第 122 页的『记录部件』

第 25 页的『Typedef』

相关参考

第 544 页的『get next』

第 818 页的『commit()』

第 830 页的『rollback()』

维护 EGL 代码

对 EGL 源代码进行行注释

要注释一行代码，执行下列操作：

1. 单击该行，然后单击鼠标右键。将显示上下文菜单。
2. 选择**注释**。注释指示符（//）位于该行的开头。

要注释多个连续的代码行，执行下列操作：

1. 单击起始行。按住鼠标左键，将光标放在结尾行上。释放鼠标按钮，将突出显示行的范围。
2. 单击鼠标右键，然后从上下文菜单中选择**注释**。注释指示符（//）位于所选范围的行的开头。

使用同一过程来对代码行取消注释，但从上下文菜单中选择**取消注释**。

相关任务

第 118 页的『创建 EGL 源文件』

第 252 页的『在 .egl 文件中打开部件』

相关参考

第 441 页的『EGL 编辑器』

搜索部件

如果文件已在 EGL 编辑器中打开，可在设置搜索条件后搜索部件：

1. 打开 EGL 文件。除非 EGL 编辑器处于活动状态，否则不能使用搜索工具；但是，您的搜索不会仅限于已在编辑器中打开的文件。
2. 在“工作台”菜单上单击**搜索 > EGL**。将显示“搜索”对话框。
3. 如果还未显示“EGL 搜索”选项卡，则单击**EGL 搜索**。注意，在“搜索”选项卡上指定的条件会影响结果。
4. 输入要定位的部件的名称；或者，如果要显示名称与特定字符模式相匹配的部件的列表，请在名称中嵌入通配符：
 - 问号（?）表示一个任意字符
 - 星号（*）表示一串任意字符

例如，输入 *myForm?Group* 将找到名为 *myForm1Group* 和 *myForm2Group* 的部件，但找不到 *myForm10Group*。输入 *myForm*Group* 将找到名为 *myForm1Group*、*myForm2Group* 和 *myForm10Group* 的部件。

5. 要让搜索区分大小写（以区分 *myFormGroup* 与 *MYFORMGROUP*），单击该复选框。
6. 在“搜索”框中，选择一种部件类型，或选择**任何元素**以向所有部件类型展开搜索。
7. 在“限制范围”框中，选择将搜索范围限制为部件声明和 / 或部件首选项的选项。

8. 在“作用域”框中，选择**工作空间**以搜索工作空间，选择**包括的项目**以搜索“项目资源管理器”中当前突出显示的项目，或选择**工作集**以搜索定义的项目集合。如果选择“工作集”作用域，则单击**选择**按钮以选择现有工作集或定义新的工作集。
9. 单击**搜索**按钮。搜索的结果将显示在“搜索”视图中。
10. 如果在“搜索”视图中双击某个文件，该文件将在 EGL 编辑器中打开，并且匹配部件将突出显示。如果文件中有多个匹配项，第一个匹配项将突出显示。

编辑器的左页边距中的箭头指示每个匹配部件的位置。

相关概念

第 16 页的『部件』

相关任务

『在 .egl 文件中打开部件』

相关参考

第 441 页的『EGL 编辑器』

查看部件引用

可显示程序、库、PageHandler 或报告处理程序部件中引用的 EGL 部件的分层视图；并且可以访问这些部件：

1. 以下列其中一种方式打开“部件引用”视图：
 - 在“项目资源管理器”中，右键单击包含程序、库、PageHandler 或报告处理程序部件的 EGL 文件。选择**在部件引用中打开**。
 - 或者，在 EGL 编辑器中打开 EGL 文件：
 - a. 如果未显示“大纲”视图，则通过从“窗口”菜单中选择**显示视图 > 大纲**来打开该视图。
 - b. 在“大纲”视图中，右键单击文件并单击**在部件引用中打开**。
2. 程序、库、PageHandler 或报告处理程序部件位于层次结构的最顶层；每个被引用部件都是该层次结构中的子项；对于每个部件，视图都会显示相应的参数、数据声明、使用声明和功能。
3. 双击某个部件。相关的源文件会在 EGL 编辑器中打开，并且部件名会突出显示。

相关概念

第 13 页的『EGL 项目、包和文件』

第 16 页的『部件』

相关任务

第 253 页的『在“项目资源管理器”中定位 EGL 源文件』

『在 .egl 文件中打开部件』

相关参考

第 441 页的『EGL 编辑器』

在 .egl 文件中打开部件

通过几次击键，可在工作空间中的任意位置访问构建部件之外的 EGL 部件：

1. 在工作台中，单击**浏览 > 打开部件**或者单击工具栏上的**打开部件**按钮。将显示“打开部件”对话框。
2. 输入要定位的部件的名称；或者，如果要显示名称与特定字符模式相匹配的部件的列表，请在名称中嵌入通配符：
 - 问号（?）表示一个任意字符
 - 星号（*）表示一串任意字符

例如，输入 *myForm?Group* 将找到名为 *myForm1Group* 和 *myForm2Group* 的部件，但找不到 *myForm10Group*。输入 *myForm*Group* 将找到名为 *myForm1Group*、*myForm2Group* 和 *myForm10Group* 的部件。

当您输入名称时，合格的部件将显示在“打开部件”对话框的“匹配的部件”部分中。

3. 从部件列表中，选择要打开的部件。对话框的“限定符”部分将显示一个路径，该路径包含用于存放所选部件的文件夹、项目、包和源文件。在多个部件同名的情况下，可以通过单击要打开的文件的**路径**来选择部件。
4. 单击**确定**。这就在 EGL 编辑器中打开了包含所选部件的源文件，并突出显示了部件名。

相关概念

第 13 页的『EGL 项目、包和文件』

第 16 页的『部件』

相关任务

第 118 页的『创建 EGL 源文件』

『在“项目资源管理器”中定位 EGL 源文件』

相关参考

第 441 页的『EGL 编辑器』

在“项目资源管理器”中定位 EGL 源文件

如果您正在编辑 EGL 源文件，则可以在“项目资源管理器”视图中快速定位该文件。“在项目资源管理器中显示”上下文菜单选项执行下列操作：

- 打开“项目资源管理器”视图，如果该视图尚未打开的话
- 展开定位源文件所需的“项目资源管理器”树节点
- 突出显示源文件

要在“项目资源管理器”中定位 EGL 源文件，执行下列操作：

1. 在打开的 EGL 源文件的编辑器区域内单击鼠标右键。将显示上下文菜单。
2. 从上下文菜单中选择**在项目资源管理器中显示**。

相关任务

第 118 页的『创建 EGL 源文件』

第 252 页的『在 .egl 文件中打开部件』

相关参考

第 441 页的『EGL 编辑器』

在“项目资源管理器”中删除 EGL 文件

要在“项目资源管理器”中删除 EGL 文件，执行下列操作：

1. 单击 EGL 文件并按 Delete 键。或者，右键单击 EGL 文件，在显示上下文菜单时选择删除。
2. 系统将要求您确认您想要删除该文件。单击是删除该文件或单击否以取消删除。

相关任务

第 118 页的『创建 EGL 源文件』

第 253 页的『在“项目资源管理器”中定位 EGL 源文件』

调试 EGL 代码

EGL 调试器

在工作台中，EGL 调试器允许您进行 EGL 代码调试，而不要求首先生成输出。下列类别有效：

- 要调试 PageHandler 以及在 J2EE 上下文中使用的程序，可以按照调试方式使用本地 WebSphere Application Server 测试环境：
 - 对于 Web 应用程序中的所有在 J2EE 下运行的代码，都必须使用该环境。
 - 对于在 J2EE 下的批处理应用程序中运行的程序，可以使用该环境。
- 要调试其它代码（不在 J2EE 下运行的批处理应用程序；或者文本应用程序），请使用位于 WebSphere 测试环境外部的启动配置。在这种情况下，只需几次击键就可以启动调试会话。

如果您正在开发准备部署在 J2EE 上下文中的批处理程序，则可以使用启动配置来在非 J2EE 上下文中调试程序。尽管设置较为简单，但是需要调整一些值：

- 在使用启动配置时，需要将构建描述符选项 J2EE 的值设置为 NO。
- 并且，需要根据关系数据库访问方式的差别来调整 Java 属性值：
 - 对于 J2EE，指定类似于 *jdbc/MyDB* 的字符串，该字符串是在 JNDI 注册表中与数据源绑定的名称。可以按照下列方式指定该字符串：
 - 通过设置构建描述符选项 sqlJNDIName；或者
 - 通过在“EGL SQL 数据库连接”首选项页的“连接 JNDI 名称”字段中指定一个值；有关详细信息，请参阅设置 *SQL 数据库连接首选项*。
 - 对于非 J2EE，指定类似于 *jdbc:db2:MyDB* 的连接 URL。可以按照下列方式指定该字符串：
 - 通过设置构建描述符选项 sqlDB；或者
 - 通过在“EGL SQL 数据库连接”首选项页的“连接 URL”字段中指定一个值；有关详细信息，请参阅设置 *SQL 数据库连接首选项*。

下面有一节描述了构建描述符与 EGL 首选项之间进行的交互。

调试器命令

使用下列命令来与 EGL 调试器进行交互：

添加断点

标识一行，处理将在该位置暂停。当代码执行暂停时，您可以检查变量值以及文件和屏幕的状态。

除非除去断点，否则会将断点从一个调试会话记录到下一个调试会话。

不能在空行或注释行上设置断点。

禁用断点

释放断点，但不将其除去。

启用断点

激活先前被禁用的断点。

除去断点

清除断点，以使处理不再自动地在该行暂停。

除去全部断点

清除每个断点。

运行

运行代码，直到下一个断点或者直到运行单元结束为止。（在任何一种情况下，调试器都在主函数中的第一个语句处停止。）

运行至行

运行所有语句，直到（但不包括）指定行上的语句。

单步跳入

运行下一个 EGL 语句并暂停。

以下列表指示了当对特定语句类型发出**单步跳入**命令时发生的情况：

call

如果被调用程序在 EGL 调试器中运行，则停止在被调用程序的第一个语句处。
如果被调用程序在 EGL 调试器外部运行，则停止在当前程序中的下一个语句处。

EGL 调试器将在工作台中的每个项目中搜索接收程序。

converse

等待用户输入。该输入导致处理在下一个运行语句处停止，该语句可能位于验证器函数中。

forward

如果代码前进到 PageHandler，则调试器将等待用户输入并在下一个运行语句处停止，该语句可能位于验证器函数中。

如果代码前进到程序中，则调试器在该程序中的第一个语句处停止。

函数调用

在函数中的第一个语句处停止。

JavaLib.invoke 和相关函数

停止在下一个 Java 语句处，以便可以调试由 Java 访问函数提供的 Java 代码。

show, transfer

在接收控制权的程序的第一个语句处停止。目标程序是在 EGL 调试器中运行的 EGL 源代码，而不是 EGL 生成的代码。

在 **show** 语句或具有 *transfer to a transaction* 格式的 **transfer** 语句之后，EGL 调试器切换至新程序的构建描述符或者（如果未使用任何此类构建描述符的话）提示用户输入新的构建描述符。新程序可以具有与先前运行的程序不同的属性集。

EGL 调试器将在工作台中的每个项目中搜索接收程序。

单步跳过

运行下一个 EGL 语句并暂停，但不在当前函数所调用的函数中停止。

以下列表指示了当对特定语句类型发出**单步跳过**命令时发生的情况：

converse

等待用户输入，然后跳过任何验证函数（除非存在有效的断点）。在 **converse** 语句后面的语句处停止。

forward

如果代码前进到 PageHandler，则调试器将等待用户输入并在下一个运行语句处停止，但是，除非存在有效的断点，否则不在验证器函数中停止。

如果代码前进到程序中，则调试器在该程序中的第一个语句处停止。

show, transfer

在接收控制权的程序的第一个语句处停止。目标程序是在 EGL 调试器中运行的 EGL 源代码，而不是 EGL 生成的代码。

在 **show** 语句或具有 *transfer to a transaction* 格式的 **transfer** 语句之后，EGL 调试器切换至新程序的构建描述符或者（如果未使用任何此类构建描述符的话）提示用户输入新的构建描述符。新程序可以具有与先前运行的程序不同的属性集。

EGL 调试器将在工作台中的每个项目中搜索接收程序。

单步返回

运行要返回至调用程序或函数所需的语句；然后，在该程序或函数中接收控制权的语句处暂停。

如果在验证器函数中发出**单步返回**命令，则会导致异常。在该情况下，该行为等同于**单步跳入**命令的行为，后者主要表示 EGL 调试器运行下一个语句并暂停。

EGL 调试器将下列 EGL 语句视为空运算符：

- **sysLib.audit**
- **sysLib.purge**
- **sysLib.startTransaction**

例如，可以在这些语句处添加断点，但**单步跳入**命令仅仅是前进到下一个语句，而没有其它作用。

最后，如果对函数中运行的最后一个语句发出**单步跳入**或**单步跳过**命令（并且，如果该语句不是 **return**、**exit program** 或 **exit stack**），则处理在函数本身中暂停，以便您可以查看该函数的局部变量。在这种情况下，要使调试会话继续，请发出另一个命令。

构建描述符的使用

构建描述符有助于确定调试环境的各个方面。EGL 调试器根据下列规则选择构建描述符：

- 如果为程序或 PageHandler 指定了调试构建描述符，则 EGL 调试器使用该构建描述符。有关如何建立调试构建描述符的详细信息，请参阅设置**设置**构建描述符。
- 如果未指定调试构建描述符，则 EGL 调试器将提示您从构建描述符列表中进行选择或接受 **None** 值。如果接受 **None** 值，则 EGL 调试器将构造构建描述符以便在调试会话期间使用；首选项确定了 VisualAge Generator 兼容性是否起作用。
- 如果指定 **None** 或者指定的构建描述符缺少某些必需的数据库连接信息，则 EGL 调试器通过查看首选项来获取连接信息。有关如何设置那些首选项的详细信息，请参阅设置 *SQL 数据库连接* 首选项。

如果您正在调试将要在 Java 环境中的文本或批处理应用程序中使用的程序，并且该程序发出一个 **transfer** 语句，该语句将控制权切换至也将要在 Java 环境中的另一个运行单元中使用的程序，则 EGL 调试器使用对接收程序指定的构建描述符。对构建描述符的选择基于前面描述的规则。

如果您正在调试被另一个程序调用的程序，则 EGL 调试器使用对被调用程序指定的构建描述符。对构建描述符的选择基于上面描述的规则，但是，如果未指定构建描述符，当调用被调用程序时调试器不会提示您输入构建描述符；而是继续使用调用程序的构建描述符。

注：如果调用程序和被调用程序二者中的一个（而不是全部）利用 VisualAge Generator 兼容性，则必须对它们使用不同的构建描述符。VisualAge 兼容性的生成时状态是由构建描述符选项 **VAGCompatibility** 的值确定的。

用于调试代码的构建描述符或资源关联部件可能与用于生成代码的构建描述符或资源关联部件不同。

SQL 数据库访问

为了确定用来访问 SQL 数据库的用户标识和密码，EGL 调试器按顺序考虑下列源，直到找到信息或考虑了每个源为止：

1. 在调试时使用的构建描述符；特别是，构建描述符选项 **sqlIID** 和 **sqlPassword**。
2. SQL 首选项页，如设置 **SQL 数据库连接首选项**中所述；在该页上，也可以指定其它连接信息。
3. 在连接时显示的交互式对话框。仅当选择了**根据需要提示输入 SQL 用户标识和密码**复选框时才会显示这样的对话框。

call 语句

如上所述，EGL 调试器通过解释 EGL 源代码来响应 **transfer** 或 **show** 语句。然而，EGL 调试器通过查看构建描述符中指定的链接选项部件（如果有的话）来响应 **call** 语句。如果引用的链接选项部件包含调用的 **callLink** 元素，则结果如下所示：

- 如果将 **callLink** 属性 **remoteComType** 设置为 **DEBUG**，则 EGL 调试器解释 EGL 源代码。调试器通过引用 **callLink** 属性 **package** 和 **location** 来查找源。
- 如果 **callLink** 属性 **remoteComType** 未设置为 **DEBUG**，则调试器将调用 EGL 生成的代码并使用链接选项部件中的信息，就象是调试器正在运行 EGL 生成的 Java 程序一样。

当不存在链接信息时，EGL 调试器通过解释 EGL 源代码来响应 **call** 语句。在下列情况下，没有链接信息：

- 未使用构建描述符；或者
- 使用了构建描述符，但在该构建描述符中未指定链接选项部件；或者
- 在构建描述符中指定了链接选项部件，但引用的部件不具有引用被调用程序的 **callLink** 元素。

如果调试器运行 EGL 源代码，则可以通过从调用程序中发出**单步跳入**命令来运行该程序中的语句。但是，如果调试器调用生成的代码，则调试器运行整个程序；**单步跳入**命令的工作方式与**单步跳过**命令相似。

在调试时使用的系统类型

`sysVar.systemType` 包含系统类型的值。并且，如果请求了与 VisualAge Generator 的开发时兼容性，则 `VGLib.getVAGSysType` 提供了另一个值。

`sysLib.systemType` 中的值与构建描述符选项 `system` 的值相同，但是，在下列两种情况下，值为 `DEBUG`：

- 您选择首选项将 `systemType` 设置为 `DEBUG`，如设置 *EGL 调试器首选项* 中所述；或者
- 您指定了 `NONE` 来作为在调试会话期间要使用的构建描述符，而不考虑该首选项的值。

系统函数 `VGLib.getVAGSysType` 返回 `sysLib.systemType` 值的 VisualAge Generator 等效值；有关详细信息，请参阅 *VGLib.getVAGSysType* 中的表。

EGL 调试器端口

EGL 调试器使用一个端口来建立与 Eclipse 工作台的通信。缺省端口号是 8345。如果另一个应用程序正在使用该端口，或者如果该端口被防火墙阻塞，则设置另一个值，如设置 *EGL 调试器首选项* 中所述。

如果指定了除 8345 以外的值来作为 EGL 调试器端口，并且如果将在 J2EE 服务器上调试 EGL 程序，则必须编辑服务器配置：

1. 转至“环境”选项卡的“系统属性”部分
2. 单击“添加”
3. 对于“名称”，输入 `com.ibm.debug.egl.port`
4. 对于“值”，输入新端口号

建议

当您准备使用 EGL 调试器时，请考虑下列建议（这些建议中的绝大部分假定在调试代码时 `sysVar.systemType` 设置为 `DEBUG`）：

- 如果正在从数据库中检索日期，但期望运行时代码以不同于 ISO 格式的格式来检索该日期，则请编写一个函数来转换日期，但仅当系统类型为 `DEBUG` 时才调用该函数。ISO 格式为 `yyyy-mm-dd`，这是唯一可供调试器使用的格式。
- 要指定外部 Java 类以便在调试器运行时使用，请修改类路径，如设置 *EGL 调试器首选项* 中所述。可能需要额外的类，以便支持（例如）MQSeries、JDBC 驱动程序或 Java 访问函数。
- 当调试已由 JSF（而不是由另一个 EGL 函数）调用的 `PageHandler` 函数时，使用“运行”来离开该函数，而不是使用“单步跳过”、“单步跳入”或“单步返回”。使用那三个“单步”命令中的任何一个都将转到 `PageHandler` 的生成的 Java 代码，这在调试 EGL 时没什么用处。如果使用其中一个“单步”命令，请使用“运行”来离开生成的 Java 代码并在浏览器中显示 Web 页面。
- 如果正在使用 SQL 选项 `WITH HOLD`（或 EGL 的等效选项），则您需要知道，选项 `WITH HOLD` 不可用于 EGL 生成的 Java，在 EGL 调试器中也不可用。将落实语句置于只在运行时才被调用的条件语句中，也许可以部分地解决该局限性，如下示例所示：

```
if (systemType not debug)
  sysLib.commit();
end
```

如果在 J2EE 服务器上调试 EGL 程序，或者通过 EGL 侦听器来调试它，则必须配置服务器或 EGL 侦听器以指示 EGL 调试器端口号：

- 要配置 J2EE 服务器，请编辑服务器配置：
 1. 转至“环境”选项卡的“系统属性”部分
 2. 单击**添加**
 3. 对于**名称**，输入 *com.ibm.debug.egl.port*
 4. 对于**值**，输入新端口号
- 要配置 EGL 侦听器，请编辑 EGL 侦听器的启动配置：
 1. 转至“自变量”选项卡
 2. 在 **VM 自变量** 字段中，输入以下内容：

```
-Dcom.ibm.debug.egl.port=portNumber
```

```
portNumber
```

新端口号

相关概念

第 400 页的『与 VisualAge Generator 的兼容性』

第 107 页的『EGL 调试器的字符编码选项』

第 241 页的『VSAM 支持』

相关任务

第 110 页的『设置 SQL 数据库连接首选项』

第 106 页的『为 EGL 调试器设置首选项』

第 108 页的『设置缺省构建描述符』

相关参考

第 381 页的『callLink 元素中的 remoteComType』

第 364 页的『sqlDB』

第 364 页的『sqlID』

第 365 页的『sqlJNDIName』

第 366 页的『sqlPassword』

第 842 页的『getVAGSysType()』

第 859 页的『systemType』

调试 J2EE 之外的应用程序

在 EGL 调试器中启动非 J2EE 应用程序

要在 EGL 调试会话中开始调试 EGL 文本程序或非 J2EE 基本程序，启动配置是必需的。启动配置定义程序的文件位置并指定应该如何启动程序。您可以让 EGL 应用程序创建启动配置（隐式创建），也可以自己创建启动配置（请参阅在 *EGL 调试器中创建启动配置*）。

要使用隐式创建的启动配置来启动程序，执行下列操作：

1. 在“项目资源管理器”视图中，右键单击要启动的 EGL 源文件。另外，如果已在 EGL 编辑器中打开了 EGL 源文件，则可以在“大纲”视图中右键单击程序。
2. 将显示上下文菜单。
3. 单击**调试 EGL 程序**。将创建一个启动配置，并且将在 EGL 调试器中启动程序。

要查看隐式创建的启动配置，执行下列操作：

1. 单击工具栏上“调试”按钮旁边的箭头。将显示上下文菜单。
2. 单击**调试**。将显示“调试”对话框。启动配置的名称将显示在“名称”字段中。隐式创建的启动配置是根据项目和源文件名进行命名的。

注：也可以通过单击“运行”菜单中的**调试**来显示“调试”对话框。

相关概念

第 255 页的『EGL 调试器』

相关任务

『在 EGL 调试器中创建启动配置』

第 265 页的『在 EGL 调试器中单步执行应用程序』

第 264 页的『在 EGL 调试器中使用断点』

第 266 页的『在 EGL 调试器中查看变量』

在 EGL 调试器中创建启动配置

要在 EGL 调试会话中开始调试 EGL 文本程序或非 J2EE 基本程序，启动配置是必需的。启动配置定义应该如何启动程序。您可以创建启动配置（显式创建），也可以让 EGL 应用程序为您创建启动配置（请参阅在 *EGL 调试器中启动非 J2EE 程序*）。

要使用显式创建的启动配置来启动程序，执行下列操作：

1. 单击工具栏上“调试”按钮旁边的箭头，然后单击**调试**，或者从“运行”菜单中选择**调试**。
2. 将显示“调试”对话框。
3. 单击“配置”列表中的 **EGL 程序**，然后单击**新建**。
4. 如果未在“项目资源管理器”视图中突出显示 EGL 源文件，则启动配置将被命名为 *New_configuration*。如果已在“项目资源管理器”视图中突出显示了 EGL 源文件，则启动配置将与该 EGL 源文件同名。如果要更改启动配置的名称，则在“名称”字段中输入新名称。
5. 如果“装入”选项卡的“项目”字段中的名称不正确，则单击**浏览**。将显示项目列表。单击项目，然后单击**确定**。
6. 如果 EGL 程序源文件中的名称不正确或者该字段是空的，则单击**搜索**。将显示 EGL 源文件列表。单击源文件，然后单击**确定**。
7. 如果对“调试”对话框上的任何字段作了更改，则单击**应用**以保存启动配置设置。
8. 单击**调试**以在 EGL 调试器中启动该程序。

注：如果尚未使用**应用**来保存启动配置设置，则单击**还原**将除去您所作的所有更改。

相关概念

第 255 页的『EGL 调试器』

相关任务

第 260 页的『在 EGL 调试器中启动非 J2EE 应用程序』

第 265 页的『在 EGL 调试器中单步执行应用程序』

第 264 页的『在 EGL 调试器中使用断点』

第 266 页的『在 EGL 调试器中查看变量』

创建 EGL 侦听器启动配置

要调试从 EGL 生成的 Java 应用程序或包装器调用的非 J2EE EGL 应用程序，需要 EGL 侦听器启动配置。要创建 EGL 侦听器启动配置，执行下列操作：

1. 单击工具栏上“调试”按钮旁边的箭头，然后单击**调试**，或者从“运行”菜单中选择**调试**。
2. 将显示“调试”对话框。
3. 单击“配置”列表中的 **EGL 侦听器**，然后单击**新建**。
4. 侦听器启动配置名为 *New_configuration*。如果要更改启动配置的名称，则在“名称”字段中输入新名称。
5. 如果不输入端口号，则端口号缺省为 8346；否则输入端口号。每个 EGL 侦听器都需要它自己的端口。
6. 单击**应用**以保存侦听器启动配置。
7. 单击**调试**以启动 EGL 侦听器。

相关概念

第 255 页的『EGL 调试器』

相关任务

第 261 页的『在 EGL 调试器中创建启动配置』

第 260 页的『在 EGL 调试器中启动非 J2EE 应用程序』

第 265 页的『在 EGL 调试器中单步执行应用程序』

第 264 页的『在 EGL 调试器中使用断点』

第 266 页的『在 EGL 调试器中查看变量』

调试 J2EE 应用程序

为进行 EGL Web 调试而准备服务器

要调试在 WebSphere Application Server 中运行的 EGL Web 程序，必须为进行调试而准备服务器。必须对每个服务器执行一次准备步骤，并且不需要再次执行该步骤，即使关闭了工作台亦如此。

要为进行调试而准备服务器，执行下列操作：

1. 如果要使用 WebSphere V5.1 测试环境，则确保该服务器已停止。如果要使用 WebSphere Application Server V6.0，则确保该服务器正在运行。之所以这样做，其原因是 V6.0 代码是正在起作用的服务器。
2. 在“服务器”视图中，右键单击服务器。将显示上下文菜单。
3. 选择**启用/禁用 EGL 调试**。出现一条消息，指示您已经禁用了 EGL 调试。
4. 如果要调试生成的 Java 而不是 EGL，则再次右键单击服务器并选择**启用/禁用 EGL 调试**。出现一条消息，指示您已经禁用了 EGL 调试。

相关概念

第 255 页的『EGL 调试器』

第 309 页的『WebSphere Application Server 和 EGL』第 171 页的『Web 支持』

相关任务

『启动 EGL Web 调试会话』

『为进行 EGL Web 调试而启动服务器』

第 265 页的『在 EGL 调试器中单步执行应用程序』

第 264 页的『在 EGL 调试器中使用断点』

第 266 页的『在 EGL 调试器中查看变量』

为进行 EGL Web 调试而启动服务器

如果要使用基于 EGL 的 Web 应用程序来访问 JNDI 数据源，则只有先前配置了 Web 应用程序服务器时，才能遵循当前主题中的指示信息。有关特定于 WebSphere 的后台信息，请参阅 *WebSphere Application Server* 和 *EGL*。

并且，如果希望调试 EGL Web 程序，则必须按准备服务器以便进行 *EGL Web* 调试中的描述准备服务器。

要启动服务器以进行调试，则执行下列操作：

1. 在“服务器”视图中，右键单击服务器
2. 选择**调试 > 在服务器上调试**。

相关概念

第 255 页的『EGL 调试器』

第 309 页的『WebSphere Application Server 和 EGL』第 171 页的『Web 支持』

相关任务

第 262 页的『为进行 EGL Web 调试而准备服务器』

『启动 EGL Web 调试会话』

第 265 页的『在 EGL 调试器中单步执行应用程序』

第 264 页的『在 EGL 调试器中使用断点』

第 266 页的『在 EGL 调试器中查看变量』

启动 EGL Web 调试会话

如果要使用基于 EGL 的 Web 应用程序来访问 JNDI 数据源，则只有先前配置了 Web 应用程序服务器时，才能遵循当前主题中的指示信息。有关特定于 WebSphere 的后台信息，请参阅 *WebSphere Application Server* 和 *EGL*。

并且，如果希望调试 EGL Web 程序，则必须按准备服务器以便进行 *EGL Web* 调试中的描述准备服务器。如果已经启动了服务器以进行调试（如启动服务器以进行 *EGL Web* 调试中所述），则可以节省当前过程中的时间。

要启动 EGL Web 调试会话，执行下列操作：

1. 在“项目资源管理器”中，展开 **WebContent** 和 **WEB-INF** 文件夹。右键单击想要运行的 JSP 文件，然后选择**调试 > 在服务器上调试**。将显示“选择服务器”对话框。

2. 如果已经为此 Web 项目配置了服务器，选择**选择现有服务器**，然后从列表中选择服务器。单击**完成**以启动服务器（如果必要的话），将应用程序部署至服务器并启动该应用程序。
3. 如果尚未为此 Web 项目配置服务器，可按如下所示进行处理，但仅当应用程序未访问 JNDI 数据源时才能这样做：
 - a. 选择**手工定义服务器**。
 - b. 指定主机名，即（对于本地机器）**localhost**。
 - c. 选择一种服务器类型，该类型与您打算在运行时在其上部署应用程序的 Web 应用程序服务器类似。选择包括 **WebSphere V5.1 测试环境**和 **WebSphere V6.0 服务器**。
 - d. 如果在处理当前项目时不打算更改您的选择，则选择**将服务器设置为缺省项目服务器**复选框。
 - e. 在大多数情况下，可避免此步骤；但如果您希望指定不同于缺省值的设置，则单击**下一步**并进行选择。
 - f. 单击**完成**以启动服务器，将应用程序部署至服务器并启动该应用程序。

相关概念

第 255 页的『EGL 调试器』

第 309 页的『WebSphere Application Server 和 EGL』第 171 页的『Web 支持』

相关任务

第 262 页的『为进行 EGL Web 调试而准备服务器』

第 263 页的『为进行 EGL Web 调试而启动服务器』

第 265 页的『在 EGL 调试器中单步执行应用程序』

『在 EGL 调试器中使用断点』

第 266 页的『在 EGL 调试器中查看变量』

在 EGL 调试器中使用断点

断点用来暂停程序的执行。可以在 EGL 调试会话之内或之外管理断点。使用断点时，请记住以下几点：

- “源代码”视图左页边距中的蓝色标记指示已设置并启用断点。
- “源代码”视图左页边距中的白色标记指示已设置断点，但已将该断点禁用。
- 左页边距中没有标记表示未设置断点。

添加或删除断点

通过执行下列其中一项操作来在 EGL 源文件中添加或删除单个断点：

- 将光标定位在“源代码”视图左页边距中的断点行上并双击。
- 将光标定位在“源代码”视图左页边距中的断点行上并单击鼠标右键。将显示上下文菜单。单击适当的菜单项。

禁用或启用断点

通过执行下列操作来禁用或启用 EGL 源文件中的单个断点：

1. 在“断点”视图中，右键单击断点。将显示上下文菜单。
2. 单击适当的菜单项。

除去所有断点

通过执行下列操作来除去 EGL 源文件中的所有断点:

1. 右键单击“断点”视图中显示的任何断点。将显示上下文菜单。
2. 单击**全部除去**。

相关概念

第 255 页的『EGL 调试器』

相关任务

- 第 261 页的『在 EGL 调试器中创建启动配置』
- 第 260 页的『在 EGL 调试器中启动非 J2EE 应用程序』
- 『在 EGL 调试器中单步执行应用程序』
- 第 266 页的『在 EGL 调试器中查看变量』

在 EGL 调试器中单步执行应用程序

如 *EGL 调试器* 中所述, EGL 调试器提供了下列命令来在调试会话期间控制程序的执行:

继续 运行代码, 直到遇到下一个断点或直到程序结束为止。

运行至行

允许在“源代码”视图中选择可执行行并运行代码至该行。

单步跳入

运行下一个 EGL 语句并暂停。程序在被调用函数的第一个语句处停止。

单步跳过

运行下一个 EGL 语句并暂停, 但不在当前函数所调用的函数中停止。

单步返回

返回至调用程序或函数。

除“运行至行”以外, 可以通过下列方法访问每个命令:

- 单击“调试”视图的工具栏上的适当按钮; 或者
- 单击“运行”菜单中的适当菜单项; 或者
- 右键单击“调试”视图中突出显示的线程, 然后单击适当的菜单项。

要使用“运行至行”, 请在程序暂停时执行如下操作:

1. 将光标定位在“源代码”视图左页边距的可执行行上, 然后单击鼠标右键。将显示上下文菜单。
2. 单击**运行至行**。

使用“运行至行”时, 请记住以下几点:

- “调试”视图或“运行”菜单没有提供该操作
- “运行至行”将停止在已启用的断点上

相关概念

第 255 页的『EGL 调试器』

相关任务

- 第 261 页的『在 EGL 调试器中创建启动配置』
- 第 260 页的『在 EGL 调试器中启动非 J2EE 应用程序』
- 第 264 页的『在 EGL 调试器中使用断点』
- 『在 EGL 调试器中查看变量』

在 EGL 调试器中查看变量

每当程序暂停时，您可以查看程序变量的当前值。

要查看程序的变量，执行下列操作：

1. 在“变量”视图中，展开导航器中的部件以查看它们的变量。
2. 要显示变量的类型，请单击工具栏上的**显示类型名**按钮。
3. 要在单独的窗格中显示变量详细信息，请单击该变量，然后单击工具栏上的**显示详细信息**按钮。

相关概念

- 第 255 页的『EGL 调试器』

相关任务

- 第 261 页的『在 EGL 调试器中创建启动配置』
- 第 260 页的『在 EGL 调试器中启动非 J2EE 应用程序』
- 第 265 页的『在 EGL 调试器中单步执行应用程序』
- 第 264 页的『在 EGL 调试器中使用断点』

使用 EGL 构建部件

创建构建文件

要创建构建文件，执行下列操作：

1. 标识用来包含文件的项目或文件夹。如果还没有项目或文件夹，则必须创建项目或文件夹。项目应该是 EGL 或 EGL Web 项目。
2. 在工作台中，单击文件 > 新建 > **EGL 构建文件**。
3. 选择将包含 EGL 构建文件的项目或文件夹。在“文件名”字段中，输入 EGL 构建文件的名称，例如，MyEGLbuildParts。该文件名必须具有扩展名 .eglbld。如果没有指定扩展名或者指定了无效的扩展名，则自动将扩展名追加到文件名的末尾。
4. 单击**完成**以创建不带 EGL 构建部件声明的构建文件。构建文件出现在“项目资源管理器”视图中，并在 EGL 构建部件编辑器中自动打开。
5. 要在创建构建文件之前添加 EGL 构建部件，请单击**下一步**。选择要添加的构建部件的类型，然后单击**下一步**。输入构建部件的名称和描述，然后单击**完成**。构建文件出现在“项目资源管理器”视图中，并在 EGL 构建部件编辑器中自动打开。

相关概念

第 13 页的『EGL 项目、包和文件』

第 1 页的『EGL 简介』

相关任务

第 271 页的『将构建描述符部件添加至 EGL 构建文件』

第 284 页的『将链接选项部件添加至 EGL 构建文件』

第 289 页的『将 import 语句添加至 EGL 构建文件』

第 280 页的『将资源关联部件添加至 EGL 构建文件』

第 116 页的『创建 EGL Web 项目』

设置一般构建选项

构建描述符部件

构建描述符部件控制生成过程。此部件包含若干种信息：

- **构建描述符选项**指定如何生成和准备 EGL 输出，并且一部分构建描述符选项可导致将其它构建部件包括在生成过程中。有关特定选项的详细信息，请参阅**构建描述符选项**。
- **Java 运行时属性**将值赋给下列属性：
 - `vgj.datemask.gregorian.long.locale`，它包含在下列两种情况的任何一种情况下使用的日期掩码：
 - 为系统变量 `VGVar.currentFormattedGregorianCalendar` 生成的 Java 代码被调用；或者

- EGL 验证长度为 10 或更长的页项或文本表单字段（如果项属性 **dateFormat** 设置为 *systemGregorianCalendarFormat* 的话）。

本节的末尾讨论了 *locale* 的含义。

- *vgj.datemask.gregorian.short.locale*，它包含一个日期掩码，如果项属性 **dateFormat** 设置为 *systemGregorianCalendarFormat*，则 EGL 验证页项或文本表单字段的长度是否小于 10 时将使用此日期掩码。

本节的末尾讨论了 *locale* 的含义。

- *vgj.datemask.julian.long.locale*，它包含在下列两种情况的任何一种情况下使用的日期掩码：
 - 调用为系统变量 *VGVar.currentFormattedJulianDate* 生成的 Java 代码；或者
 - 如果项属性 **dateFormat** 设置为 *systemJulianDateFormat*，则 EGL 验证页项或文本表单字段的长度是否为 8 或更长。

本节的末尾讨论了 *locale* 的含义。

- *vgj.datemask.julian.short.locale*，它包含一个日期掩码，如果项属性 **dateFormat** 设置为 *systemJulianDateFormat*，则 EGL 验证页项或文本表单字段的长度是否小于 10 时将使用此日期掩码。

本节的末尾讨论了 *locale* 的含义。

- *vgj.jdbc.database.SN*，它标识可供 Java 代码使用的数据库。

在部署时，当指定 *SN* 的替换值时，必须定制属性本身的名称。而替换值又必须与包含在 *VGLib.connectionService* 调用中的服务器名称或包含在 *sysLib.connect* 调用中的数据库名称相匹配。

还必须定制日期掩码属性的名称：

- 在给定的运行单元中，每个最初有效的属性都具有一个名称，此名称的最后一个限定符（*locale*）与程序属性 **vgj.nls.code** 中的值相匹配
- 在 Web 应用程序中，如果程序设置系统变量 *sysLib.setLocale*，则另一组属性生效

主构建描述符： 系统管理员可能要求您使用主构建描述符来指定不能被覆盖并且对于在 EGL 安装中发生的每一次生成都起作用的信息。根据主构建描述符中描述的机制，系统管理员按名称及包含部件的 EGL 构建文件来标识该部件。

如果主构建描述符中的信息对于特定生成过程不足够，或者如果未标识任何主构建描述符，则可以在生成时指定构建描述符以及包含特定于生成的部件的 EGL 构建文件。特定于生成的构建描述符（与主构建描述符相似）必须位于 EGL 构建文件的顶层。

可以根据特定于生成的构建描述符创建构建描述符链，以便处理链中的第一个构建描述符之后才处理第二个构建描述符，处理第二个之后才处理第三个，等等。当定义给定的构建描述符时，通过将值赋给构建描述符选项 **nextBuildDescriptor** 来开始一个链（或继续一个链）。系统管理员可以使用同一技术来根据主构建描述符创建链。将在后面链接信息的含义进行描述。

根据“对部件的引用”中描述的规则，构建描述符引用的任何构建部件对于引用构建描述符都必须是可视的。例如，构建部件可以是链接选项部件或资源关联部件，也可以是下一个构建描述符。

选项的优先顺序: 对于给定的构建描述符选项（或 Java 运行时属性），在生成时最初处理的值仍然有效，并且总体优先顺序如下所示：

1. 主构建描述符
2. 特定于生成的构建描述符，后跟自其扩展的链
3. 从主构建描述符扩展的链

此方案的优点在于：

- 系统管理员可以方便地通过设置主构建描述符来指定不更改的值。
- 可以使用特定于生成的构建描述符来方便地指定特定于生成的值。
- 项目经理可以通过定制一个或多个构建描述符来方便地指定一组缺省值。在大多数这样的情况下，特定于生成的构建描述符指向由项目经理开发的链中的第一个构建描述符。

当组织开发一组必须以相似方式生成和准备的程序时，缺省选项会很有用。

- 尽管不经常使用此功能部件，系统管理员还是可以通过建立从主构建描述符扩展的链来创建一组常规缺省值。

如果多次使用了给定的构建描述符，则只有对该构建描述符的第一次访问才有效。另外，对于特定的选项，只有第一次指定才有效。

示例: 让我们假定主构建描述符包含下列（非实际）“选项 - 值”对：

```
OptionX          02
OptionY          05
```

在此示例中，特定于生成的构建描述符（名为 myGen）包含下列“选项 - 值”对。

```
OptionA          20
OptionB          30
OptionC          40
OptionX          50
```

如在 myGen 中所标识的那样，下一个构建描述符是 myNext01，它包含下列内容：

```
OptionA          120
OptionD          150
```

如在 myNext01 中所标识的那样，下一个构建描述符是 myNext02，它包含下列内容：

```
OptionB          220
OptionD          260
OptionE          270
```

如在主构建描述符中所标识的那样，下一个构建描述符是 myNext99，它包含以下内容：

```
OptionZ          99
```

EGL 按以下顺序接受选项值：

1. 主构建描述符中的选项值：

```
OptionX          02
OptionY          05
```

那些选项将覆盖所有其它选项。

2. 特定于生成的构建描述符 myGen 中的值：

OptionA	20
OptionB	30
OptionC	40

忽略了 myGen 中的 optionX 的值。

3. myNext01 和 myNext02 中其它选项的值:

OptionD	150
OptionE	270

忽略了 myNext01 中的 optionA 的值，也忽略了 myNext02 中的 optionD 的值。

4. myNext99 中其它选项的值:

OptionZ	99
---------	----

相关概念

第 295 页的『构建』

第 315 页的『Java 运行时属性』

第 20 页的『对部件的引用』

『主构建描述符』

第 16 页的『部件』

相关任务

第 271 页的『将构建描述符部件添加至 EGL 构建文件』

第 280 页的『将资源关联部件添加至 EGL 构建文件』

第 272 页的『编辑构建描述符中的一般选项』

第 275 页的『编辑构建描述符中的 Java 运行时属性』

第 281 页的『编辑 EGL 构建文件中的资源关联部件』

相关参考

第 347 页的『构建描述符选项』

第 493 页的『Java 运行时属性（详细信息）』

第 820 页的『connect()』

第 839 页的『connectionService()』

第 832 页的『setLocale()』

第 863 页的『currentFormattedGregorianCalendar』

第 864 页的『currentFormattedJulianDate』

主构建描述符

安装可以为构建选项提供它自己的一组缺省值并控制那些缺省值是否可以被覆盖。

要设置主构建描述符，请在同一个构建文件中创建两个构建描述符部件，第一个构建描述符部件使用构建描述符选项 **nextBuildDescriptor** 来引用第二个构建描述符部件。第一个部件中的选项对不能被覆盖的选项指定缺省值。第二个部件中的选项对可以被覆盖的选项指定缺省值。

要在工作台中安装主构建描述符，请将类似于以下的插件 XML 文件添加到工作台插件目录中:

```
<?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 = "filePath.buildFileName"
name = "masterBuildPartName" />
</extension>
</plugin>

```

文件路径 (*filePath*) 与工作空间目录有关。

如果正在使用 EGL SDK，则在名为 `eglmaster.properties` 的文件中声明主构建描述符的名称和文件路径名。此文件必须位于 `CLASSPATH` 环境变量中列示的目录中。属性文件的格式如下所示：

```

masterBuildDescriptorName=masterBuildPartName
masterBuildDescriptorFile=fullyQualifiedPathforEGLBuildFile

```

相关概念

- 第 295 页的『构建』
- 第 267 页的『构建描述符部件』
- 第 297 页的『构建规划』
- 第 13 页的『EGL 项目、包和文件』

相关任务

- 『将构建描述符部件添加至 EGL 构建文件』

相关参考

- 第 347 页的『构建描述符选项』
- 第 448 页的『`eglmaster.properties` 文件的格式』
- 第 463 页的『主构建描述符 `plugin.xml` 文件的格式』

将构建描述符部件添加至 EGL 构建文件

构建描述符部件控制生成过程。它包含选项名和它们的相关值，这些选项与值对指定如何生成和准备 EGL 输出。某些选项指定生成过程中的其它控制部件，例如，生成过程中的资源关联部件。可以将构建描述符部件添加至 EGL 构建文件。有关更多信息，请参阅 *构建描述符部件*。要添加构建描述符部件，执行下列操作：

1. 要使用 EGL 构建部件编辑器打开 EGL 构建文件，在“项目资源管理器”中执行下列操作：
 - a. 右键单击 EGL 构建文件
 - b. 选择打开方式 > **EGL 构建部件编辑器**。
2. 如果未显示“大纲”视图，则通过从“窗口”菜单中选择 **显示视图 > 大纲** 来打开该视图。
3. 在“大纲”视图中，右键单击构建文件，然后单击 **添加部件**。
4. 单击 **构建描述符** 单选按钮，然后单击 **下一步**。
5. 为构建描述符选择符合 EGL 部件名约定的名称。在“名称”字段中，输入构建描述符的名称。
6. 在“描述”字段中输入对构建部件的描述。
7. 单击 **完成**。这就在 EGL 构建文件中声明了构建描述符，并在 EGL 构建部件编辑器中显示了构建描述符一般选项。

8. (可选)可以创建构建描述符链,以便在处理了链中的第一个构建描述符之后才处理第二个构建描述符,处理第二个构建描述符之后才处理第三个构建描述符。如果要开始或继续构建描述符链,则在“选项”列表的 **nextBuildDescriptor** 选项字段中指定下一个构建描述符。要填充 **nextBuildDescriptor** 选项字段,执行下列操作:
 - a. 使用“选项”列表上的滚动条来向下滚动,直到看到 **nextBuildDescriptor** 选项为止。
 - b. 如果 **nextBuildDescriptor** 行未突出显示,则单击一次以选择行。
 - c. 单击一次“值”字段以使该字段处于编辑模式。
 - d. 可以在“值”字段中输入下一个构建描述符的名称,或者从下拉列表中选择现有的构建描述符。

相关概念

第 267 页的『构建描述符部件』

相关任务

『编辑构建描述符中的一般选项』

第 275 页的『编辑构建描述符中的 Java 运行时属性』

第 276 页的『从 EGL 构建文件中除去构建描述符部件』

相关参考

第 346 页的『EGL 构建文件格式』

第 612 页的『命名约定』

编辑构建描述符中的一般选项

构建描述符部件控制生成过程。要编辑一般构建描述符选项和符号参数,执行下列操作:

1. 要使用 EGL 构建部件编辑器打开 EGL 构建文件,在“项目资源管理器”中执行下列操作:
 - a. 右键单击 EGL 构建文件
 - b. 选择打开方式 > **EGL 构建部件编辑器**
2. 如果未显示“大纲”视图,则通过从“窗口”菜单中选择**显示视图 > 大纲**来打开该视图。
3. 在“大纲”视图中,右键单击构建描述符并选择**打开**。编辑器视图的右上角有两个按钮。确保按了“显示常规构建描述符选项”按钮(两个按钮中的第一个按钮)。EGL 构建部件编辑器将显示当前部件定义的一般构建描述符选项。
4. (可选)可以创建构建描述符链,以便在处理了链中的第一个构建描述符之后才处理第二个构建描述符,处理第二个构建描述符之后才处理第三个构建描述符。如果要开始或继续构建描述符链,则在 **nextBuildDescriptor** 字段中指定下一个构建描述符。如果 **nextBuildDescriptor** 行未突出显示,则单击一次以选择行,然后单击一次“值”字段以使该字段处于编辑模式。可以在“值”字段中输入下一个构建描述符的名称,或者从下拉列表中选择现有的构建描述符。
5. 要指定 EGL 输出的生成和准备,从**构建选项过滤器**下拉列表中选择选项值对的分组。如果要定义的选项未突出显示,则单击一次以选择行,然后单击一次“值”字段以使该字段处于编辑模式。可以输入选项值,或者,如果下拉列表可用的话,就可以选择现有的值。如果要将选项与值对的视图限制为已经定义的对,则单击“只显示已指定的选项”复选框。

相关概念

第 267 页的『构建描述符部件』

相关任务

第 271 页的『将构建描述符部件添加至 EGL 构建文件』

第 275 页的『编辑构建描述符中的 Java 运行时属性』

第 276 页的『从 EGL 构建文件中除去构建描述符部件』

相关参考

第 346 页的『EGL 构建文件格式』

选择 Java 生成选项

构建描述符选项是在构建描述符部件中设置的。要选择构建描述符选项，请启动 EGL 编辑器并编辑构建描述符部件。

当您开始从图形用户界面中编辑构建描述符部件时，EGL 编辑器中包含一个窗格，该窗格列示了所有 EGL 构建描述符选项。要进行限制以便只显示适用于程序的选项，请从“构建选项过滤器”下拉菜单中选择类别。

选择您想要的每个选项，并设置它的值。该值可以是文字、符号或者是文字与符号的组合。可以在 EGL 部件编辑器中定义符号参数；有关详细信息，请参阅编辑一般构建描述符选项。

两个构建描述符选项 - **genDirectory** 和 **destDirectory** - 允许您对值或值的一部分使用符号参数。例如，对于 **genDirectory** 的值，可以指定 `C:\genout\%EZEENV%`。因此，如果为 Windows 环境生成，则实际生成目录为 `C:\genout\WIN`。

不需要指定所列示的所有选项。如果不为一个构建描述符选项指定值，则当该选项适用于生成上下文时，就将使用该选项的缺省值。

如果已经指定了一个主构建描述符，则该构建描述符中的选项值将覆盖所有其它构建描述符中的值。当生成时，主构建描述符和生成构建描述符可以链接至其它构建描述符，如构建描述符部件中所述。

相关概念

第 267 页的『构建描述符部件』

相关任务

第 272 页的『编辑构建描述符中的一般选项』

『生成 Java 包装器』

相关参考

第 347 页的『构建描述符选项』

生成 Java 包装器

生成相关程序时，可以生成 Java 包装器类。有关如何设置构建描述符的详细信息，请参阅 *Java 包装器*。

相关概念

第 293 页的『生成』

第 293 页的『将 Java 代码生成到项目中』第 274 页的『Java 包装器』

相关任务

第 296 页的『构建 EGL 输出』

第 304 页的『处理生成到目录中的 Java 代码』

相关参考

第 347 页的『构建描述符选项』

第 502 页的『Java 包装器类』

第 615 页的『Java 包装器生成输出』

Java 包装器: Java 包装器是一组类，这些类可充当下列可执行文件之间的接口：

- 一方为 servlet 或手工编写的 Java 程序
- 另一方为生成的程序或 EJB 会话 bean

如果使用具有下列特征的构建描述符，则会生成 Java 包装器类：

- 构建描述符选项 **enableJavaWrapperGen** 设置为 **yes** 或 **only**；并且
- 构建描述符选项 **linkage** 引用一个链接选项部件，后者包含 **callLink** 元素来指引从包装器到程序的调用；并且
- 出现下列两种情况中的一种：
 - 从包装器到程序的调用是通过 EJB 会话 bean 进行的（在这种情况下，**callLink** 元素的 **linkType** 属性设置为 **ejbCall**）；或者
 - 从包装器到程序的调用是远程的（在这种情况下，**callLink** 元素的 **type** 属性设置为 **remoteCall**）；并且，**callLink** 元素的 **javaWrapper** 属性设置为 **yes**。

如果在 Java 包装器类与 EGL 生成的程序之间存在 EJB 会话 bean，则在使用具有下列特征的构建描述符时，将生成 EJB 会话：

- 构建描述符选项 **enableJavaWrapperGen** 设置为 **yes** 或 **only**；并且
- 构建描述符选项 **linkage** 引用一个链接选项部件，后者包含 **callLink** 元素来指引从包装器到 EJB 会话 bean 的调用（在这种情况下，**callLink** 元素的 **type** 属性设置为 **ejbCall**）。

有关使用类的更多详细信息，请参阅 *Java 包装器类*。有关类名的详细信息，请参阅生成的输出（参考）。

相关概念

第 483 页的『生成的输出』

第 297 页的『Java 程序、PageHandler 和库』

第 8 页的『运行时配置』

相关任务

第 273 页的『生成 Java 包装器』

相关参考

第 484 页的『生成的输出（参考）』

第 502 页的『Java 包装器类』

第 615 页的『Java 包装器生成输出』

编辑构建描述符中的 Java 运行时属性

当编辑构建描述符部件时，可以对下列 Java 运行时属性赋值，在 *Java 运行时属性*（详细信息）中对这些属性作了详细说明：

- `vgj.jdbc.database.SN`
- `vgj.datemask.gregorian.long.locale`
- `vgj.datemask.gregorian.short.locale`
- `vgj.datemask.julian.long.locale`
- `vgj.datemask.julian.short.locale`

仅当生成 Java 代码时才会使用您的赋值。

要编辑属性，执行下列操作：

1. 要使用 EGL 构建部件编辑器打开 EGL 构建文件，在“项目资源管理器”中执行下列操作：
 - a. 右键单击 EGL 构建文件
 - b. 选择打开方式 > **EGL 构建部件编辑器**
2. 如果未显示“大纲”视图，则通过从“窗口”菜单中选择**显示视图 > 大纲**来打开该视图。
3. 在“大纲”视图中，右键单击构建描述符并选择**打开**。EGL 部件编辑器将显示当前部件定义的一般构建描述符选项。
4. 单击编辑器工具栏上的**显示 Java 运行时属性**按钮。
5. 要添加 Java 运行时属性 **vgj.jdbc.database.SN**，执行下列操作：
 - a. 在标题为“连接的数据库映射”的屏幕区域中，单击**添加**按钮
 - b. 输入您在编写系统字 `VGLib.connectionService` 时使用的“服务器名称”；在生成的属性的名称中，此值替换 `SN`
 - c. 如果“连接的数据库映射”列表中的行未突出显示，则单击一次以选择行，然后单击一次 `JNDI` 名称或 `URL` 字段以使该字段处于编辑模式。输入一个值，它对于 `J2EE` 连接和非 `J2EE` 连接有不同的含义：
 - 对于 `J2EE` 连接（在生产环境中需要此连接），该值是在 `JNDI` 注册表中与数据源绑定的名称；例如，`jdbc/MyDB`
 - 对于标准 `JDBC` 连接（可能用于调试），该值是连接 `URL`；例如，`jdbc:db2:MyDB`
6. 要指定当您编写 `VGVar.currentFormattedGregorianCalendarDate`（对于格里历日期）或 `VGVar.currentFormattedJulianDate`（对于儒略历日期）时使用的日期掩码，或者当 EGL 验证长度为 10 或更长的页项或文本表单字段并且 `dateFormat` 属性为 `systemGregorianCalendarDateFormat` 或 `systemJulianDateFormat` 时使用的日期掩码，执行下列操作：
 - a. 在标题为“日期掩码”的屏幕区域中，单击**添加**按钮
 - b. 在“语言环境”列中，选择列表框中的一个代码；选择的值将替换先前日期掩码属性中所列出的 `locale`。在运行时只使用其中一个条目：其 `locale` 的值与 Java 运行时属性 `vgj.nls.code` 的值相匹配的条目
 - c. 如果“日期掩码”列表中的行未突出显示，则单击一次以选择行，然后单击一次“长格里历掩码”字段以使该字段处于编辑模式。从列表框中选择掩码或者输入掩码；可以将除了 `D`、`Y` 或数字之外的字符用作分隔符，并且缺省值是特定于语言环境的

- d. 如果“日期掩码”列表中的行未突出显示，则单击一次以选择行，然后单击一次“长儒略历掩码”字段以使该字段处于编辑模式。从列表框中选择掩码或者输入掩码；可以将除了 D、Y 或数字之外的字符用作分隔符，并且缺省值是特定于语言环境的
7. 要指定当 EGL 验证长度小于 10 的页项或文本表单字段并且 **dateFormat** 属性为 *systemGregorianCalendar* 或 *systemJulianDate* 时使用的日期掩码，执行下列操作：
 - a. 在标题为“日期掩码”的屏幕区域中，单击**添加**按钮
 - b. 在“语言环境”列中，选择列表框中的一个代码；选择的值将替换先前日期掩码属性中所列出的 **locale**。在运行时只使用其中一个条目：其 **locale** 的值与 Java 运行时属性 **vgj.nls.code** 的值相匹配的条目
 - c. 如果“日期掩码”列表中的行未突出显示，则单击一次以选择行，然后单击一次“短格里历掩码”字段以使该字段处于编辑模式。从列表框中选择掩码或者输入掩码；可以将除了 D、Y 或数字之外的字符用作分隔符，并且缺省值是特定于语言环境的
 - d. 如果“日期掩码”列表中的行未突出显示，则单击一次以选择行，然后单击一次“短儒略历掩码”字段以使该字段处于编辑模式。从列表框中选择掩码或者输入掩码；可以将除了 D、Y 或数字之外的字符用作分隔符，并且缺省值是特定于语言环境的
 8. 要除去一个赋值，单击它，然后单击**除去**按钮。

相关概念

- 第 267 页的『构建描述符部件』
- 第 315 页的『Java 运行时属性』

相关任务

- 第 271 页的『将构建描述符部件添加至 EGL 构建文件』
- 第 272 页的『编辑构建描述符中的一般选项』
- 『从 EGL 构建文件中除去构建描述符部件』

相关参考

- 第 346 页的『EGL 构建文件格式』
- 第 493 页的『Java 运行时属性（详细信息）』
- 第 839 页的『connectionService()』
- 第 863 页的『currentFormattedGregorianCalendar』
- 第 864 页的『currentFormattedJulianDate』

从 EGL 构建文件中除去构建描述符部件

要从 EGL 构建描述符文件中除去构建描述符部件，执行下列操作：

1. 要使用 EGL 构建部件编辑器打开 EGL 构建文件，在“项目资源管理器”中执行下列操作：
 - a. 右键单击 EGL 构建文件
 - b. 选择打开方式 > **EGL 构建部件编辑器**
2. 如果未显示“大纲”视图，则通过从“窗口”菜单中选择**显示视图** > **大纲**来打开该视图
3. 在“大纲”视图中，右键单击构建描述符部件并单击**除去**

相关概念

第 267 页的『构建描述符部件』

相关任务

第 271 页的『将构建描述符部件添加至 EGL 构建文件』

第 272 页的『编辑构建描述符中的一般选项』

第 275 页的『编辑构建描述符中的 Java 运行时属性』

设置外部文件、打印机和队列关联

资源关联和文件类型

访问外部文件、打印机或队列的 EGL 固定记录具有逻辑文件或队列名。（对于打印机，在大多数运行时系统上，逻辑文件名是 *printer*。）此名称的长度不能超过 8 个字符，并且它的意义仅在于作为一种将记录与系统名称相关的方法（目标系统使用系统名称来访问物理文件、打印机或队列）。

对于文件或队列，系统名称的缺省值是文件或队列名。对于打印机，不存在缺省值。

您可以执行下列两个操作的其中一个或两个，而不接受缺省值：

- 在生成时，使用构建描述符来控制生成过程，而该描述符又引用特定资源关联部件。资源关联部件使文件名与要部署生成的代码的目标平台上的系统名称相关。
- 在运行时（在大多数情况下），可以更改特定于记录的变量 `resourceAssociation`（对于文件或队列）中的值或系统变量 `ConverseVar.printerAssociation`（对于打印输出）中的值。您的目的是覆盖在缺省情况下指定的系统名称或通过指定资源关联部件指定的系统名称。

资源关联部件不适用于下列记录类型：

- `basicRecord`，这是因为基本记录不与数据存储进行交互
- `SQLRecord`，这是因为 SQL 记录与关系数据库进行交互

资源关联部件： 资源关联部件是一组关联元素，每一个关联元素都具有下列特征：

- 特定于逻辑文件或队列名
- 具有一组条目，每个条目都特定于目标系统；每个条目都标识目标平台上的文件类型和系统名称，在某些情况下还提供其它信息

可以将关联元素看作具有分层关系的一组属性和值，如下示例所示：

```
// an association element
property: fileName
value:    myFile01

// an entry, with multiple properties
property: system
value:    aix
property: fileType
value:    spool
property: systemName
value:    employee

// a second entry
property: system
value:    win
```

```

property: fileType
value: seqws
property: systemName
value: c:\myProduct\myFile.txt

```

在此示例中，文件名 `myFile01` 与下列文件相关：

- AIX 上的 `employee`
- Windows 2000/NT/XP 上的 `myFile.txt`

文件名必须是一个有效名称、一个星号或者以有效名称开头后跟一个星号。星号是等同于一个或多个字符的通配符，它可以标识一组名称。例如，包含以下文件名值的关联元素等同于以 `myFile` 开头的任何文件名：

```
myFile*
```

如果有多个元素对于在程序中使用的文件名有效，则 EGL 使用适用的第一个元素。例如，一系列关联元素可依次由文件名的下列值来表征：

```

myFile
myFile*
*

```

考虑与最后一个值相关联的元素，在这里，`myFile` 的值只是一个星号。这样的元素可适用于任何文件；但是对于特定文件，仅当前面的元素不适用时，最后一个元素才适用。例如，如果程序引用 `myFile01`，则在第二个元素中指定的链接将取代第三个元素来定义如何处理引用。

在生成时，EGL 选择特定的关联元素以及适当的第一个条目。一个条目在下面任何一种情况下都是适当的：

- 在生成操作的目标系统与 **system** 属性之间存在匹配；或者
- **system** 属性具有以下值：

```
any
```

例如，如果正在为 AIX 进行生成操作，则 EGL 使用第一个引用 **aix** 或 **any** 的条目。

文件类型： 文件类型确定关联元素中的给定条目需要哪些属性。下表描述了 EGL 文件类型。

文件类型	描述
ibmcobol	EGL 生成的 Java 程序以远程方式访问的 VSAM 文件。有关在这种情况下指定系统名称的详细信息，请参阅 VSAM 支持。
mq	一个 MQSeries 消息队列；有关如何使用这样的队列的详细信息，请参阅 MQSeries 支持。
seqws	由 EGL 生成的 Java 程序访问的串行文件。
spool	AIX 或 Linux 上的假脱机文件。

记录类型和 VSAM： 在三种类型的固定记录中，每一种都适合于用来访问 VSAM 数据集，但仅当记录的关联元素中的文件类型是 `ibmcobol`、`vsam` 或 `vsamrs` 时才如此：

- 如果固定记录为 `indexedRecord` 类型，则 VSAM 数据集是具有主索引或备用索引的“关键字顺序数据集”
- 如果固定记录为 `relativeRecord` 类型，则 VSAM 数据集是“相对记录数据集”

- 如果固定记录为 `serialRecord` 类型，则 VSAM 数据集是“输入顺序数据集”

有关更多详细信息： 有关资源关联的更多详细信息，请参阅下列主题：

- [记录和文件类型交叉引用](#)
- [关联元素](#)

相关概念

- 第 123 页的『固定记录部件』
- 第 242 页的『MQSeries 支持』
- 第 16 页的『部件』
- 第 125 页的『记录类型和属性』
- 第 122 页的『记录部件』
- 第 241 页的『VSAM 支持』

相关任务

- 第 280 页的『将资源关联部件添加至 EGL 构建文件』
- 第 281 页的『编辑 EGL 构建文件中的资源关联部件』
- 第 282 页的『从 EGL 构建文件中除去资源关联部件』

相关参考

- 第 340 页的『关联元素』
- 第 673 页的『记录和文件类型交叉引用』
- 第 786 页的『`recordName.resourceAssociation`』
- 第 361 页的『`resourceAssociations`』
- 第 367 页的『`system`』
- 第 846 页的『`printerAssociation`』

逻辑工作单元

当更改类别为不可恢复的资源（如 Windows 2000 上的串行文件）时，您的工作相对持久；无论是代码还是 EGL 运行时服务都不能简单地废除那些更改。当更改类别为可恢复的资源（如关系数据库）时，代码或 EGL 运行时服务可以落实更改来使工作持久，也可以回滚更改以返回到上次落实更改时有效的内容。

可恢复的资源如下所示：

- 关系数据库
- 配置为可恢复的 CICS 队列和文件
- MQSeries 消息队列（除非 MQSeries 记录另有指定），如 *MQSeries 支持* 中所述

逻辑工作单元用来标识成组落实或回滚的输入操作。当代码更改可恢复的资源时，工作单元开始；当发生第一个下列事件时，工作单元结束：

- 代码调用系统函数 `sysLib.commit` 或 `sysLib.rollback` 来落实或回滚更改
- EGL 运行时服务执行回滚以响应代码中没有处理的硬错误；在这种情况下，会从内存中除去运行单元中的所有程序
- 发生隐式落实，隐式落实在下列情况下发生：
 - 程序发出 `show` 语句。
 - 运行单元中的顶级程序成功地结束，如运行单元中所述。
 - 显示一个 Web 页面，这在 PageHandler 发出 `forward` 语句时发生。

- 程序发出 **converse** 语句，并且存在下列任何一种情况：
 - 未处于 VisualAge Generator 兼容性方式，并且程序是分段程序
 - **ConverseVar.commitOnConverse** 设置为 1
 - 处于 VisualAge Generator 兼容方式，并且 **ConverseVar.segmentedMode** 设置为 1

Java 工作单元： 在 Java 运行单元中，详细信息如下所示：

- 当任何 Java 程序由于硬错误而结束时，其效果等同于执行回滚、关闭游标并释放锁。
- 当运行单元成功结束时，EGL 就会执行落实、关闭游标并释放锁。
- 可以使用多个连接来从多个数据库进行读取，但是，由于只有一阶段落实可用，所以在一个工作单元中只应该更新一个数据库。有关相关信息，请参阅 *VGLib.connectionService*。
- 当通过 EGL 生成的 EJB 会话 bean 来访问 EGL 生成的程序时，事务控制可能受事务属性（也称为容器事务类型）的影响，该属性位于 EJB 会话 bean 的部署描述符中。仅当调用的链接选项部件的 callLink 元素的属性 **remoteComType** 是 direct 时，事务属性才会影响事务控制，如 callLink 元素中的 *remoteComType* 所述。

EJB 会话 bean 在生成时具有事务属性 REQUIRED，但可以在部署时更改该值。有关事务属性的含义的详细信息，请参阅 Java 文档。

相关概念

第 242 页的『MQSeries 支持』

第 678 页的『运行单元』

第 209 页的『SQL 支持』

相关任务

第 328 页的『设置 J2EE JDBC 连接』

第 240 页的『了解如何建立标准 JDBC 连接』

相关参考

第 230 页的『缺省数据库』

第 818 页的『commit()』

第 839 页的『connectionService()』

第 830 页的『rollback()』

第 502 页的『Java 包装器类』

第 377 页的『callLink 元素中的 luvControl』

第 381 页的『callLink 元素中的 remoteComType』

第 364 页的『sqlDB』

将资源关联部件添加至 EGL 构建文件

资源关联部件使文件名与想要部署生成代码的目标平台上的系统资源名相关。可以将资源关联部件添加至 EGL 构建文件。有关详细信息，请参阅[资源关联和文件类型](#)。要添加资源关联部件，执行下列操作：

1. 要使用 EGL 构建部件编辑器打开 EGL 构建文件，在“项目资源管理器”中执行下列操作：
 - a. 右键单击 EGL 构建文件
 - b. 选择打开方式 > **EGL 构建部件编辑器**

2. 如果未显示“大纲”视图，则通过从“窗口”菜单中选择**显示视图 > 大纲**来打开该视图。
3. 在“大纲”视图中，右键单击构建文件，然后单击**添加部件**。
4. 单击**资源关联**，然后单击**下一步**。
5. 为资源关联部件选择符合 EGL 部件名称约定的名称。在“名称”字段中，输入资源关联部件的名称。
6. 在“描述”字段中输入对部件的描述。
7. 单击**完成**。这将把资源关联部件添加至 EGL 构建文件，并在 EGL 构建部件编辑器中打开资源关联部件页。

相关概念

第 267 页的『构建描述符部件』

第 277 页的『资源关联和文件类型』

相关任务

『编辑 EGL 构建文件中的资源关联部件』

第 282 页的『从 EGL 构建文件中除去资源关联部件』

相关参考

第 346 页的『EGL 构建文件格式』

第 612 页的『命名约定』

编辑 EGL 构建文件中的资源关联部件

资源关联部件使文件名与想要部署生成代码的目标平台上的系统资源名相关。

要编辑资源关联部件，执行下列操作：

1. 要使用 EGL 构建部件编辑器打开 EGL 构建文件，在“项目资源管理器”中执行下列操作：
 - a. 右键单击 EGL 构建文件
 - b. 选择**打开方式 > EGL 构建部件编辑器**
2. 如果未显示“大纲”视图，则通过从“窗口”菜单中选择**显示视图 > 大纲**来打开该视图。
3. 在“大纲”视图中，右键单击资源关联部件，并单击**打开**。部件编辑器将显示当前部件定义。
4. 要对部件添加新的“关联”元素，单击**添加关联**或者按 **Insert** 键，然后输入逻辑文件名或选择逻辑文件名。
5. 要更改与逻辑文件名相关联的缺省系统名称，可以执行下列任何一项操作：
 - 在“关联元素”列表中选择相应的行，然后单击一次该名称，以使该字段处于编辑模式。从“系统”下拉列表中选择新的系统名称。
 - 在“所选系统条目的属性”列表中，单击一次 **system** 属性，以使与该属性相关联的“值”字段处于编辑模式。从“值”下拉列表中选择新的系统名称。
6. 要更改与逻辑文件名相关联的缺省文件类型，可以执行下列任何一项操作：
 - 在“关联元素”列表中选择与逻辑文件名相对应的行，然后单击一次该名称，以使该字段处于编辑模式。从“文件类型”下拉列表中选择新的文件类型。
 - 在“关联元素”列表中选择与逻辑文件名相对应的行。在“所选系统条目的属性”列表中，单击一次 **fileType** 属性，以使与该属性相关联的“值”字段处于编辑模式。从“值”下拉列表中选择文件类型。

7. 根据需要来修改资源关联。

- 要使多个系统和多组相关属性与一个逻辑文件名相关联，在“关联元素”列表中选择与逻辑文件名相对应的行。在“关联元素”列表的底部，单击**添加系统**。现在，已选择了所添加的行，并且可以编辑这些行。
- 要从相关联的逻辑文件名中除去系统和相关属性，在“关联元素”列表中选择与逻辑文件名相对应的行。在“关联元素”列表的底部，单击**除去**或者按 **Delete** 键。
- 要除去逻辑文件名和任何相关联的系统，在“关联元素”列表中选择与逻辑文件名相对应的行。在“关联元素”列表的底部，单击**除去**或者按 **Delete** 键。

相关概念

第 267 页的『构建描述符部件』

第 277 页的『资源关联和文件类型』

相关任务

第 280 页的『将资源关联部件添加至 EGL 构建文件』

『从 EGL 构建文件中除去资源关联部件』

相关参考

第 346 页的『EGL 构建文件格式』

从 EGL 构建文件中除去资源关联部件

要从 EGL 构建文件中除去资源关联部件，执行下列操作：

1. 要使用 EGL 构建部件编辑器打开 EGL 构建文件，在“项目资源管理器”中执行下列操作：
 - a. 右键单击 EGL 构建文件
 - b. 选择**打开方式 > EGL 构建部件编辑器**
2. 如果未显示“大纲”视图，则通过从“窗口”菜单中选择**显示视图 > 大纲**来打开该视图
3. 在“大纲”视图中，右键单击资源关联部件并单击**除去**

相关概念

第 277 页的『资源关联和文件类型』

相关任务

第 280 页的『将资源关联部件添加至 EGL 构建文件』

第 281 页的『编辑 EGL 构建文件中的资源关联部件』

设置调用和传送选项

链接选项部件

链接选项部件指定有关下列问题的详细信息：

- 生成的程序或包装器如何调用其它生成的代码
- 生成的程序如何以异步方式转移至其它生成的程序

指定链接选项何时是最终的： 可以在两个备用项之间进行选择：

- 在生成时指定的链接选项在运行时生效；或者
- 部署时在链接属性文件中指定的链接选项在运行时生效。虽然可以手工编写该文件，但是 EGL 在以下情况下会生成它：

- 将链接选项属性 **remoteBind** 设置为 **RUNTIME**；并且
- 在将构建描述符选项 **genProperties** 设置为 **GLOBAL** 或 **PROGRAM** 的情况下生成 Java 程序或包装器。

有关使用文件的详细信息，请参阅“部署链接属性文件”。有关定制该文件的详细信息，请参阅“链接属性文件（参考）”。

链接选项部件的元素： 链接选项部件由一组元素组成，每个元素都具有一组属性和值。提供了下列类型的元素：

- **callLink** 元素指定 EGL 对给定调用使用的链接约定。

callLink 元素始终适用于被调用程序。下列关系有效：

- 如果 **callLink** 元素引用您要生成的程序，则该元素将帮助确定是否生成 Java 包装器，该包装器允许从 Java 代码访问该程序；有关概述，请参阅 *Java 包装器*。如果您指示 Java 包装器通过 EJB 会话 bean 来访问程序，则 **callLink** 元素还将导致生成 EJB 会话 bean。
- 如果要生成 Java 程序，并且如果 **callLink** 元素引用由该程序调用的程序，则 **callLink** 元素指定如何实现该调用；例如，该调用是本地的还是远程的。如果您指示调用 Java 程序通过 EJB 会话 bean 进行调用，则 **callLink** 元素将导致生成 EJB 会话 bean。
- **asynchLink** 元素指定生成的程序如何以异步方式转移至另一个程序（这种转移是当转移程序调用系统函数 `sysLib.startTransaction` 时发生的）。
- **transferToProgram** 元素指定生成的 COBOL 程序如何将控制权转移至程序以及结束处理。此元素并非用于 Java 输出，仅对发出类型为 *transfer to program* 的 **transfer** 语句的主程序有意义。
- **transferToTransaction** 元素指定生成的程序如何将控制权转移至事务以及结束处理。此元素仅对于发出 *transfer to transaction* 类型的 **transfer** 语句的主程序有意义。但是，当目标程序是使用 VisualAge Generator 或（当不存在别名时）EGL 生成时，该元素不是必需的。

标识元素所引用的程序或记录： 在每个元素中，属性（例如 **pgmName**）标识元素所引用的程序或记录；除非另有说明，否则该属性的值可以是有效的名称、星号或者以有效名称开头并后跟一个星号。星号是等同于一个或多个字符的通配符，它可以标识一组名称。

考虑包含下列 **pgmName** 属性值的 **callLink** 元素：

```
myProg*
```

该元素与任何以字母 *myProg* 开头的 EGL 程序部件有关。

如果有多个元素有效，则 EGL 使用第一个适用的元素。例如，一系列 **callLink** 元素可依次由下列 **pgmName** 值来表征：

```
YourProgram
YourProg*
*
```

考虑与最后一个值相关联的元素，其中 **pgmName** 的值只是一个星号。此类元素可适用于任何程序；但是对于特定的程序，仅当前面的元素不适用时，最后一个元素才适

用。例如，如果程序调用 YourProgram01，则在第二个元素（YourProg*）中指定的链接将取代第三个元素（*）来定义 EGL 如何处理调用。

在大多数情况下，具有较特殊名称的元素应该优先于具有较普通名称的那些元素。在上一个示例中，适当地定位了具有星号的元素以提供缺省链接规范。

相关概念

第 274 页的『Java 包装器』

第 16 页的『部件』

相关任务

『将链接选项部件添加至 EGL 构建文件』

第 329 页的『部署链接属性文件』

第 286 页的『编辑链接选项部件的 asynchLink 元素』

第 285 页的『编辑链接选项部件的 callLink 元素』

第 287 页的『编辑与转移相关的链接选项部件元素』

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

相关参考

第 343 页的『asynchLink 元素』

第 513 页的『call』

第 370 页的『callLink 元素』

第 360 页的『linkage』

第 598 页的『链接属性文件（详细信息）』

第 835 页的『startTransaction()』

第 589 页的『transfer』

第 873 页的『transferToTransaction 元素』

将链接选项部件添加至 EGL 构建文件

链接选项部件描述生成的 EGL 程序如何实现调用和转移以及程序如何访问文件。要添加此类部件，执行下列操作：

1. 要使用 EGL 构建部件编辑器打开 EGL 构建文件，在“项目资源管理器”中执行下列操作：
 - a. 右键单击 EGL 文件
 - b. 选择打开方式 > **EGL 构建部件编辑器**
2. 如果未显示“大纲”视图，则通过从“窗口”菜单中选择显示视图 > 大纲来打开该视图。
3. 在“大纲”视图中，右键单击构建文件，然后单击添加部件。
4. 单击链接选项，然后单击下一步。
5. 为链接选项部件选择符合 EGL 部件名约定的名称。在“名称”字段中，输入链接选项部件的名称。
6. 在“描述”字段中输入对部件的描述。
7. 单击完成。这将把链接选项部件添加至 EGL 文件，并在 EGL 构建部件编辑器中打开链接选项部件页。

相关概念

第 282 页的『链接选项部件』

相关任务

第 286 页的『编辑链接选项部件的 `asynchLink` 元素』
『编辑链接选项部件的 `callLink` 元素』

第 287 页的『编辑与转移相关的链接选项部件元素』

第 288 页的『从 EGL 构建文件中除去链接选项部件』

相关参考

第 346 页的『EGL 构建文件格式』

第 612 页的『命名约定』

编辑链接选项部件的 `callLink` 元素

链接选项部件描述生成的 EGL 程序如何实现调用和转移以及程序如何访问文件。要编辑部件的 `callLink` 元素，执行下列操作：

1. 要使用 EGL 构建部件编辑器打开 EGL 构建文件，在“项目资源管理器”中执行下列操作：
 - a. 右键单击 EGL 构建文件
 - b. 选择打开方式 > **EGL 构建部件编辑器**
2. 如果未显示“大纲”视图，则通过从“窗口”菜单中选择**显示视图 > 大纲**来打开该视图。
3. 在“大纲”视图中，右键单击链接选项部件，并单击**打开**。EGL 部件编辑器将显示当前的部件声明。
4. 单击编辑器工具栏上的“显示 `CallLink` 元素”按钮。
5. 要添加新的 `CallLink` 元素，单击**添加**或者按 `Insert` 键，然后输入程序名（`pgmName`）或从“程序名”下拉列表中选择程序名。
6. 要更改与程序名相关联的缺省调用类型，可以执行下列任一操作：
 - 在“`CallLink` 元素”列表中选择相应的行，然后单击一次“类型”字段（`localCall`、`remoteCall` 和 `ejbCall`），以使该字段处于编辑模式。从“类型”下拉列表中选择新的调用类型。
 - 在“所选 `callLink` 元素的属性”列表中，单击一次 `type` 属性，以使与该属性相关联的“值”字段处于编辑模式。从“值”下拉列表中选择新的调用类型。
7. 与程序名相关联的其它属性按调用类型列示在“所选 `callLink` 元素的属性”列表中。要更改这些属性之一的值，选择程序名。在“所选 `callLink` 元素的属性”列表中，单击一次要定义的属性，以使与该属性相关联的“值”字段处于编辑模式。通过在“值”下拉列表选择一个选项或者在“值”字段中输入新值来定义新值。对于某些属性，只能选择下拉列表中的选项。对于其它属性，只能在“值”字段中输入值。
8. 根据需要来修改“`CallLink` 元素”列表：
 - 要重新定位 `callLink` 元素，选择该元素，并单击**上移**或**下移**。
 - 要除去 `callLink` 元素，选择该元素，并单击**除去**或者按 `Delete` 键。

相关概念

第 282 页的『链接选项部件』

相关任务

第 284 页的『将链接选项部件添加至 EGL 构建文件』

『编辑链接选项部件的 asynchLink 元素』

第 287 页的『编辑与转移相关的链接选项部件元素』

第 288 页的『从 EGL 构建文件中除去链接选项部件』

相关参考

第 343 页的『asynchLink 元素』

第 370 页的『callLink 元素』

第 346 页的『EGL 构建文件格式』

第 598 页的『链接属性文件（详细信息）』

EJB 会话 bean: EJB 会话 bean 由下列组件组成:

- Home 接口，它在运行时提供对 EJB 会话 bean 的客户机访问
- 远程 bean 接口，它列示可供该客户机直接使用的方法
- Bean 实现，它包含客户机可间接使用的逻辑

EJB 会话 bean 是一个程序与另一程序之间的或 EGL Java 包装器与程序之间的中介。EJB 会话 bean 的生成很大程度上取决于在生成时使用的链接选项部件中的设置。有关详细信息，请参阅链接选项部件；特别是有关 **callLink** 元素的概述。

有关输出文件名的详细信息，请参阅生成的输出（参考）。

相关概念

第 483 页的『生成的输出』

第 282 页的『链接选项部件』

相关参考

第 484 页的『生成的输出（参考）』

编辑链接选项部件的 asynchLink 元素

链接选项部件描述生成的 EGL 程序如何实现调用和转移以及它如何访问文件。要编辑部件的 asynchLink 元素，执行下列操作:

1. 要使用 EGL 构建部件编辑器打开 EGL 构建文件，在“项目资源管理器”中执行下列操作:
 - a. 右键单击 EGL 构建文件
 - b. 选择打开方式 > **EGL 构建部件编辑器**
2. 如果未显示“大纲”视图，则通过从“窗口”菜单中选择显示视图 > 大纲来打开该视图。
3. 在“大纲”视图中，右键单击链接选项部件，并单击打开。EGL 构建部件编辑器将显示当前的部件声明。
4. 单击编辑器工具栏上的“显示 AsynchLink 元素”按钮。
5. 要添加新的 AsynchLink 元素，单击添加或者按 Insert 键，然后输入记录名（recordName）或从“记录名”下拉列表中选择记录名。
6. 要更改与记录名相关联的缺省链接类型，可以执行下列任何一项操作:

- 在“AsyncLink 元素”列表中选择相应的行，然后单击一次“类型”字段（localAsync 和 remoteAsync），以使该字段处于编辑模式。从“类型”下拉列表中选择新的链接类型。
 - 在“所选 asyncLink 元素的属性”列表中，单击一次 type 属性，以使与该属性相关联的“值”字段处于编辑模式。从“值”下拉列表中选择新的链接类型。
7. 与记录名相关联的其它属性按链接类型列示在“所选 asyncLink 元素的属性”列表中。要更改这些属性之一的值，选择记录名。在“所选 asyncLink 元素的属性”列表中，单击一次要定义的属性，以使与该属性相关联的“值”字段处于编辑模式。通过在“值”下拉列表选择一个选项或者在“值”字段中输入新值来定义新值。对于某些属性，只能选择下拉列表中的选项。对于其它属性，只能在“值”字段中输入值。
 8. 根据需要来修改“asyncLink 元素”列表：
 - 要重新定位 asyncLink 元素，选择该元素，并单击上移或下移。
 - 要除去 asyncLink 元素，选择该元素，并单击除去或者按 Delete 键。

相关概念

第 282 页的『链接选项部件』

相关任务

第 284 页的『将链接选项部件添加至 EGL 构建文件』

第 285 页的『编辑链接选项部件的 callLink 元素』

『编辑与转移相关的链接选项部件元素』

第 288 页的『从 EGL 构建文件中除去链接选项部件』

相关参考

第 343 页的『asyncLink 元素』

第 346 页的『EGL 构建文件格式』

第 598 页的『链接属性文件（详细信息）』

第 835 页的『startTransaction()』

编辑与转移相关的链接选项部件元素

链接选项部件描述生成的 EGL 程序如何实现调用和转移以及程序如何访问文件。要编辑与转移相关的部件元素，执行下列操作：

1. 要使用 EGL 构建部件编辑器打开 EGL 构建文件，在“项目资源管理器”中执行下列操作：
 - a. 右键单击 EGL 构建文件
 - b. 选择打开方式 > **EGL 构建部件编辑器**
2. 如果未显示“大纲”视图，则通过从“窗口”菜单中选择**显示视图 > 大纲**来打开该视图。
3. 在“大纲”视图中，右键单击链接选项部件，并单击**打开**。EGL 构建部件编辑器将显示当前的部件声明。
4. 单击编辑器工具栏上的“显示 TransferLink 元素”按钮。“转移到程序”和“转移到事务”列表显示。
5. 要编辑“转移到程序”列表，执行下列操作：
 - a. 在“转移到程序”列表的底部，单击**添加**或者按 **Insert** 键并输入源程序（fromPgm）名，或者从“源程序名”下拉列表中选择程序名。

- b. 要编辑目标程序 (toPgm) 名, 请在“转移到程序”列表中选择相应的行, 然后单击一次“目标程序”字段以使该字段处于编辑模式。输入程序名, 或者从“目标程序”下拉列表中选择程序名。
 - c. 如果需要别名, 则在“转移到程序”列表中选择相应的行, 然后单击一次“别名”字段以使该字段处于编辑模式。输入别名。
 - d. 要更改与程序名相关联的缺省链接类型, 请在“转移到程序”列表中选择相应的行, 然后单击一次“链接类型” (linkType) 字段以使该字段处于编辑模式。从“链接类型”下拉列表中选择新的链接类型。
6. 要编辑“转移到事务”列表, 执行下列操作:
- a. 在“转移到事务”列表的底部, 单击**添加**或者按 **Insert** 键并输入目标程序 (toPgm) 名, 或者从“目标程序名”下拉列表中选择程序名。
 - b. 如果需要别名, 则在“转移到事务”列表中选择相应的行, 然后单击一次“别名”字段以使该字段处于编辑模式。输入别名。
 - c. 要编辑与程序名相关联的“以外部方式定义”属性, 请在“转移到事务”列表中选择相应的行, 然后单击一次“以外部方式定义”字段以使该字段处于编辑模式。从“以外部方式定义”属性下拉列表中选择以外部方式定义的属性。
 - d. 根据需要来修改“转移到事务”列表:
 - 要重新定位 `transferToTransaction` 元素, 选择该元素, 并单击**上移**或**下移**。
 - 要除去 `transferToTransaction` 元素, 选择该元素, 并单击**除去**或者按 **Delete** 键。

注: 传送至程序仅与下列产品相关联:

- Rational Application Developer for iSeries
- Rational Application Developer for z/OS

相关概念

第 282 页的『链接选项部件』

相关任务

第 284 页的『将链接选项部件添加至 EGL 构建文件』

第 286 页的『编辑链接选项部件的 `asynchLink` 元素』

第 285 页的『编辑链接选项部件的 `callLink` 元素』

『从 EGL 构建文件中除去链接选项部件』

相关参考

第 346 页的『EGL 构建文件格式』

第 873 页的『`transferToTransaction` 元素』

从 EGL 构建文件中除去链接选项部件

要从 EGL 构建文件中除去链接选项部件, 执行下列操作:

1. 要使用 EGL 构建部件编辑器打开 EGL 构建文件, 在“项目资源管理器”中执行下列操作:
 - a. 右键单击 EGL 构建文件
 - b. 选择打开方式 > **EGL 构建部件编辑器**

2. 如果未显示“大纲”视图，则通过从“窗口”菜单中选择**显示视图 > 大纲**来打开该视图
3. 在“大纲”视图中，右键单击链接选项部件并单击**除去**

相关概念

第 282 页的『链接选项部件』

相关任务

第 284 页的『将链接选项部件添加至 EGL 构建文件』

第 286 页的『编辑链接选项部件的 `asynchLink` 元素』

第 285 页的『编辑链接选项部件的 `callLink` 元素』

第 287 页的『编辑与转移相关的链接选项部件元素』

设置对其它 EGL 构建文件的引用

将 `import` 语句添加至 EGL 构建文件

`Import` 语句允许 EGL 构建文件引用其它构建文件中的部件。有关导入功能部件的更多信息，请参阅导入。

要将 `import` 语句添加至 EGL 构建文件中，执行下列操作：

1. 使用 EGL 构建部件编辑器打开 EGL 构建文件。如果尚未打开文件，则在“项目资源管理器”中执行以下操作：
 - a. 在“项目资源管理器”中右键单击构建文件
 - b. 选择**打开方式 > EGL 构建部件编辑器**
2. 单击构建部件编辑器中的**导入选项卡**。
3. 单击**添加按钮**。
4. 输入或选择要导入的文件或文件夹的名称，然后单击**确定**。

相关概念

第 30 页的『`Import`』

相关任务

『编辑 EGL 构建文件中的 `import` 语句』

第 290 页的『从 EGL 构建文件中除去 `import` 语句』

编辑 EGL 构建文件中的 `import` 语句

要在 EGL 构建文件中编辑 `import` 语句，执行下列操作：

1. 使用 EGL 构建部件编辑器打开 EGL 构建文件。如果尚未打开文件，则在“项目资源管理器”中执行以下操作：
 - a. 在“项目资源管理器”中右键单击构建文件
 - b. 选择**打开方式 > EGL 构建部件编辑器**
2. 单击构建部件编辑器中的**导入选项卡**。将显示 `import` 语句。
3. 选择要更改的 `import` 语句，然后单击**编辑按钮**。
4. 输入或选择要导入的文件或文件夹的名称，然后单击**确定**。

相关概念

第 30 页的『 Import 』

相关任务

第 289 页的『将 import 语句添加至 EGL 构建文件』

『从 EGL 构建文件中除去 import 语句』

从 EGL 构建文件中除去 import 语句

要在 EGL 构建文件中除去 import 语句，执行下列操作：

1. 使用 EGL 构建部件编辑器打开 EGL 构建文件。如果尚未打开文件，则在“项目资源管理器”中执行以下操作：
 - a. 在“项目资源管理器”中右键单击构建文件
 - b. 选择打开方式 > **EGL 构建部件编辑器**
2. 单击构建部件编辑器中的导入选项卡。将显示 import 语句。
3. 选择要除去的 import 语句，然后单击**除去**按钮。

相关概念

第 30 页的『 Import 』

相关任务

第 289 页的『将 import 语句添加至 EGL 构建文件』

第 289 页的『编辑 EGL 构建文件中的 import 语句』

编辑 EGL 构建路径

有关概述材料，请参阅下列主题：

- *对部件的引用*
- *EGL 构建路径和 eglpath*

要将项目包括在 EGL 项目路径中，请执行下列步骤：

1. 在“项目资源管理器”中，右键单击要链接至其它项目的项目，然后单击**属性**。
2. 选择 **EGL 构建路径**属性页。
3. 这样就在**项目**选项卡中显示了工作空间中的所有其它项目的列表。单击要引用的每个项目旁边的复选框。
4. 要按另一种顺序放置项目或者要导出任何项目，请单击**顺序**和**导出**选项卡并执行下列操作：
 - 要更改项目在构建路径顺序中的位置，请选择该项目并单击**向上**和**向下**按钮。
 - 要导出项目，请选择相关的复选框。要一次性地处理所有项目，请单击**全部选中**或**全部不选**按钮。
5. 单击**确定**。

相关概念

第 13 页的『EGL 项目、包和文件』

第 20 页的『对部件的引用』

第 30 页的『 Import 』

第 16 页的『部件』

相关参考

第 435 页的『EGL 构建路径和 eglpath』

第 20 页的『对部件的引用』

第 30 页的『Import』

第 16 页的『部件』

生成、编译和运行 EGL 输出

生成

生成就是创建 EGL 部件的输出。

可以在工作台中从工作台批处理接口生成输出，也可以从 EGL 软件开发包（EGL SDK）生成输出。EGL SDK 提供了独立于工作台的批处理接口，该接口可以完成基于文件的生成。

生成将使用 EGL 文件的已保存版本。

相关概念

第 7 页的『开发过程』

第 483 页的『生成的输出』

相关任务

第 273 页的『生成 Java 包装器』

相关参考

第 347 页的『构建描述符选项』

第 484 页的『生成的输出（参考）』

将 Java 代码生成到项目中

如果要生成 Java 程序或包装器，则建议您（在某些情况下是必需的）设置构建描述符选项 **genProject**，此选项导致生成到项目中。

当生成到项目中时，EGL 提供了各种服务。执行下面的任务时，服务会根据项目类型的不同而有所变化：

应用程序客户机项目

当生成到应用程序客户机项目中时，EGL 执行以下操作：

- 通过将下列条目添加至项目的 Java 构建路径来提供对 EGL jar 文件（fda6.jar 和 fdaj6.jar）的准备时访问：

```
EGL_GENERATORS_PLUGINDIR/runtime/fda6.jar  
EGL_GENERATORS_PLUGINDIR/runtime/fdaj6.jar
```

有关每个条目开头部分的变量的详细信息，请参阅设置变量 *EGL_GENERATORS_PLUGINDIR*。

- 提供对 EGL jar 文件的运行时访问：
 - 将 jar 文件导入到每个引用应用程序客户机项目的企业应用程序项目中
 - 更新应用程序客户机项目中的清单，以使企业应用程序项目中的 jar 文件可用
- 将运行时值放到部署描述符中，以便可以避免从生成的 J2EE 环境文件中剪切和粘贴条目；有关此主题的概述，请参阅设置部署描述符值

接下来的任务如下所示：

1. 如果正在通过 TCP/IP 调用生成的程序，则提供对侦听器的运行时访问，如设置 *TCP/IP 侦听器* 中所述
2. 提供对非 EGL jar 文件的访问
3. 既然已经将输出文件放到项目中，您可以继续设置 J2EE 运行时环境了

EJB 项目

当生成到 EJB 项目中时，EGL 执行以下操作：

- 通过将下列条目添加至项目的 Java 构建路径来提供对 EGL jar 文件（fda6.jar 和 fdaj6.jar）的准备时访问：

```
EGL_GENERATORS_PLUGINDIR/runtime/fda6.jar
EGL_GENERATORS_PLUGINDIR/runtime/fdaj6.jar
```

有关位于每个条目开头的环境变量的详细信息，请参阅设置变量 *EGL_GENERATORS_PLUGINDIR*。

- 提供对 EGL jar 文件的运行时访问：
 - 将 fda6.jar 和 fdaj6.jar 导入到每个引用 EJB 项目的企业应用程序项目中
 - 更新 EJB 项目中的清单，以使企业应用程序项目中的 fda6.jar 和 fdaj6.jar 在运行时可用
- 自动指定 JNDI 名称，以便 EGL 运行时代码可以访问 EJB 代码；但仅当生成 EJB 会话 bean 时才执行此步骤。
- 在大多数情况下，将运行时值放到部署描述符中，以便可以避免从生成的 J2EE 环境文件中剪切和粘贴条目；有关此主题的概述，请参阅设置 *部署描述符值*。

如果 EGL 在部署描述符中找不到必需的会话元素，则 EGL 不会将运行时值放到部署描述符中。例如，当在包装器之前生成 Java 程序时，或者当构建描述符选项 *sessionBeanID* 被设置为在部署描述符中找不到的值时，就会发生这种情况。有关会话元素的详细信息，请参阅 *sessionBeanID*。

接下来的任务如下所示：

1. 提供对非 EGL jar 文件的访问
2. 生成部署代码
3. 既然已经将输出文件放到项目中，您可以继续设置 J2EE 运行时环境了

J2EE Web 项目

EGL 执行下列操作：

- 通过将 fda6.jar 和 fdaj6.jar 导入到项目的 WebContent/WEB-INF/lib 文件夹中来提供对 EGL jar 文件的访问
- 将运行时值放到部署描述符中，以便可以避免从生成的 J2EE 环境文件中剪切和粘贴条目；有关此主题的概述，请参阅设置 *部署描述符值*

接下来的任务如下所示：

1. 提供对非 EGL jar 文件的访问
2. 既然已经将输出文件放到项目中，应按对 EGL 生成的代码设置 J2EE 运行时环境中所述继续进行

Java 项目

如果要生成到非 J2EE Java 项目中以便进行调试或生产，则 EGL 执行下列操作：

- 通过将下列条目添加至项目的 Java 构建路径来提供对 EGL jar 文件（fda6.jar 和 fdaj6.jar）的访问：

```
EGL_GENERATORS_PLUGINDIR/runtime/fda6.jar
EGL_GENERATORS_PLUGINDIR/runtime/fdaj6.jar
```

有关每个条目开头部分的变量的详细信息，请参阅设置变量 *EGL_GENERATORS_PLUGINDIR*。

- 生成属性文件，但仅当构建描述符包含下列选项值时才这样做：
 - **genProperties** 设置为 GLOBAL 或 PROGRAM；并且
 - **J2EE** 设置为 NO。

如果请求全局属性文件（**rununit.properties**），EGL 将该文件放在 Java 源文件夹中，该文件夹包含 Java 包。（Java 源文件夹可能是项目中的文件夹或项目本身。）如果改为请求程序属性文件，EGL 会将该文件放在程序中。

在运行时，程序属性文件中的值用来设置标准 JDBC 连接。有关详细信息，请参阅 *了解如何建立标准 JDBC 连接*。

既然已经将输出文件放在项目中，您可以执行下列操作了：

- 如果程序访问关系数据库，则确保 Java 构建路径包含驱动程序的安装目录。例如，对于 DB2，指定包含 db2java.zip 的目录。
- 如果代码访问 MQSeries，则提供对非 EGL jar 文件的访问
- 将链接属性文件放在模块中

有关生成到非现有项目中的后果的详细信息，请参阅 *genProject*。

相关任务

第 307 页的『生成 EJB 项目的部署代码』

第 329 页的『部署链接属性文件』

第 322 页的『设置部署描述符值』

第 330 页的『提供对非 EGL jar 文件的访问』

第 307 页的『设置变量 EGL_GENERATORS_PLUGINDIR』

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

第 240 页的『了解如何建立标准 JDBC 连接』

相关参考

第 357 页的『genProject』

第 362 页的『sessionBeanID』

构建

在开发 EGL 或 EGL Web 项目时，构建一词（通常）不是指代码生成。

下列菜单选项具有特殊的含义：

构建项目

构建部分项目：

1. 验证项目中自从上次构建以来已更改的所有 EGL 文件
2. 生成自从上次生成 PageHandler 以来已更改的 PageHandler
3. 编译自从上次编译之后已更改的任何 Java 源代码

仅当未设置工作台首选项**资源修改时自动执行构建**时菜单选项**构建项目**才可用。如果已设置该首选项，则每当您保存 EGL 文件时就会执行前面描述的操作。

全部构建

执行与**构建项目**相同的操作，但对工作空间中每个打开的项目执行那些操作。

重建项目

操作如下所示：

1. 验证项目中的所有 EGL 文件
2. 生成项目中的所有 PageHandler
3. 编译自从上次编译之后已更改的任何 Java 源代码

全部重建

执行与**重建项目**相同的操作，但对工作空间中每个打开的项目执行那些操作。

当将代码生成到项目中时，在下列情况下将在本地执行 Java 编译：

- 在构建或重建项目时；或者
- 当生成源文件时；但仅当您选择了工作台首选项**在资源修改时自动执行构建**时。

当将代码生成到目录中时，EGL 将创建（可选）**构建规划**，后者是一个包含下列详细信息的 XML 文件：

- 任何将被传送到另一台机器的文件的位置；
- 通过 TCP/IP 进行的传送所需的其它信息；以及
- Java 编译语句。

在远程平台上对生成的输出进行的准备要求正在该平台上运行构建服务器。

您可能希望创建构建规划并在稍后的时间调用该规划。有关详细信息，请参阅**在生成之后调用构建规划**。

相关概念

第 267 页的『构建描述符部件』

第 297 页的『构建规划』

第 311 页的『构建服务器』

第 7 页的『开发过程』

相关任务

第 267 页的『创建构建文件』

第 303 页的『在生成之后调用构建规划』

相关参考

第 347 页的『构建描述符选项』

构建 EGL 输出

要为 Java 程序或包装器构建 EGL 输出，完成下列步骤：

1. 生成 Java 源代码到项目或目录中：

- 如果生成到项目中（建议的做法），并且如果将 Eclipse 首选项设置为在资源修改时自动进行构建，则工作台将准备输出。
 - 如果生成到目录中，则生成器的分布式构建功能将准备输出。
2. 准备生成的输出。除非您将构建描述符选项 **buildPlan** 或 **prep** 设置为 no，否则将自动完成此步骤。

相关概念

- 第 295 页的『构建』
- 第 7 页的『开发过程』
- 第 13 页的『EGL 项目、包和文件』
- 第 293 页的『将 Java 代码生成到项目中』
- 第 483 页的『生成的输出』
- 第 293 页的『生成』

相关任务

- 第 118 页的『创建 EGL 源文件』
- 第 304 页的『处理生成到目录中的 Java 代码』
- 第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

相关参考

- 第 484 页的『生成的输出（参考）』

构建规划

构建规划是一个 XML 文件，它在准备时提供下列详细信息：

- 在构建机器上需要处理哪些文件
- 处理它们时需要哪些构建脚本
- 将把输出放置在什么位置

构建规划位于开发平台上，它将所有构建步骤的情况通知构建客户机。对于每个步骤，都会向构建服务器发出请求。

除非将构建描述符选项 **buildPlan** 设置为 NO，否则，每当生成 Java 程序或包装器时，EGL 都会生成构建规划。

有关构建规划名称的详细信息，请参阅生成的输出（参考）。

相关概念

- 第 310 页的『构建脚本』
- 第 483 页的『生成的输出』

相关参考

- 第 350 页的『buildPlan』
- 第 484 页的『生成的输出（参考）』

Java 程序、PageHandler 和库

当您请求作为 Java 程序来生成程序部件时，或者当您请求生成 pageHandler 部件或与 Java 相关的库部件时，EGL 将为下列每一项生成类和文件：

- 程序部件、pageHandler 部件或库部件

- 在该部件本身中声明的或在该部件直接或间接调用的任何函数中声明的每个记录
- 所使用的每个数据表、表单组和表单

有关类名的详细信息，请参阅 *Java 程序生成输出*。

相关任务

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

相关参考

第 484 页的『生成的输出（参考）』

第 614 页的『Java 程序生成输出』

结果文件

结果文件包含有关在目标环境中完成的代码准备步骤的状态信息。仅当 EGL 尝试准备生成的输出时，您才会接收到该文件。

当生成到目录中并使用下列构建描述符选项时，将会自动进行准备：

- **prep** 设置为 YES
- **buildPlan** 设置为 YES

有关结果文件名称的详细信息，请参阅 *生成的输出（参考）*。

相关概念

第 267 页的『构建描述符部件』

第 483 页的『生成的输出』

相关任务

第 304 页的『处理生成到目录中的 Java 代码』

相关参考

第 484 页的『生成的输出（参考）』

第 350 页的『buildPlan』

第 361 页的『prep』

在工作台中生成

在工作台生成是通过生成向导或生成菜单项完成的。当您选择生成菜单项时，EGL 将使用缺省构建描述符。如果未选择缺省构建描述符，则使用生成向导。有关选择缺省构建描述符的详细信息，请参阅 *设置缺省构建描述符*。

要通过调用生成向导来在工作台中进行生成，执行下列操作：

1. 在“项目资源管理器”中右键单击资源名称（项目、文件夹或文件）。
2. 选择**通过向导生成...**选项。

生成向导包含 4 页：

1. 第一页显示要根据用户为了开始生成过程而作出的选择来生成的部件的列表。必须从该列表中至少选择一个部件之后才能继续下一个页面。界面上提供了许多按钮，以允许您选择或取消选择列表中的所有部件。

2. 第二页允许您选择要用来生成在第一个页面上选择的部件的一个或多个构建描述符。您有两种选择：
 - 从工作空间中的所有构建描述符的下拉列表中进行选择，然后使用该构建描述符来生成所有部件。
 - 为在第一个页面上选择的每个部件选择构建描述符。使用一个表来为每个部件选择构建描述符。表的第一列显示部件名；第二列显示每个部件的构建描述符的下拉列表。
3. 如果需要用户标识和密码，则第三个页面允许您为生成过程中要使用的目标机器和 SQL 数据库设置用户标识和密码。这些用户标识和密码（如果有）将覆盖在生成的每个部件的指定构建描述符中列示的用户标识和密码。您可能想在此页面上而不是在构建描述符中设置用户标识和密码，以避免将敏感信息保存在持久存储器中。
4. 第四页允许您创建可用于在工作台外部生成 EGL 程序的命令文件。在工作台批处理接口中（通过使用命令 `EGLCMD`）或者在 EGL SDK 中（通过使用命令 `EGLSDK`）可以参考该命令文件。

要创建命令文件，执行下列操作：

- a. 选择“创建命令文件”复选框
 - b. 通过输入标准路径，或者通过单击浏览并使用标准的 Windows 过程来选择文件来指定输出文件的名称
 - c. 选择或清除“自动插入 EGL 路径（`eglp`）”复选框来指定是否想要将 EGL 项目路径包括在命令文件中作为 `eglp` 的初始值；有关详细信息，请参阅 *EGL 命令文件*
 - d. 选择一个单选按钮来指示在创建命令文件时是否避免生成输出
5. 单击**完成**。

要在工作台中使用生成菜单项进行生成，执行下列其中一组步骤：

1. 在“项目资源管理器”中选择一个或多个资源名称（项目、文件夹或文件）。要选择多个资源名称，请在单击时按住 **Ctrl** 键。
2. 单击鼠标右键，然后选择**生成**菜单选项。

或者

1. 在“项目资源管理器”中双击资源名称（项目、文件夹或文件）。这就在 EGL 编辑器中打开了该文件。
2. 在编辑器窗格中单击鼠标右键，然后选择**生成**。

相关概念

第 300 页的『工作台中的生成』

相关任务

第 108 页的『设置缺省构建描述符』

相关参考

第 436 页的『`EGLCMD`』

第 446 页的『`EGLSDK`』

第 484 页的『生成的输出（参考）』

工作台中的生成

要在工作台生成输出，执行下列操作：

- 装入要生成的部件以及在生成期间要引用的任何部件。
- 选择要生成的部件。如果对文件、文件夹、包或项目调用生成过程，则 EGL 将为所选容器中的每个主部件（每个程序、PageHandler、表单组、数据表或库）创建输出。
- 启动生成操作。
- 监视进度。

为了更加方便地进行生成，建议您首先从下列类型中选择缺省构建描述符：

- 调试构建描述符（当使用 EGL 调试器时，此构建描述符适用）
- 目标系统构建描述符（用于生成部件并将它们部署在运行时环境中）

有关如何选择缺省构建描述符的详细信息，请参阅[设置缺省构建描述符](#)。

可以通过下列方法标识两个构建描述符（调试和目标系统）中的每一个：

- 在文件、文件夹、包和项目级别，作为属性
- 作为工作台首选项

特定类型的低级别构建描述符优先于同类型的任何较高级别构建描述符。例如，对当前包指定的目标系统构建描述符优先于对项目指定的目标系统构建描述符。但是，主构建描述符优先于所有其它构建描述符，如[构建描述符部件](#)中所述。

对于每个包含主部件的文件，下列优先顺序规则有效：

- 特定于文件的属性优先于所有其它属性
- 相关的文件夹属性优先于包、项目或工作台属性
- 相关的包属性优先于项目或工作台属性
- 相关的项目属性优先于工作台属性
- 如果除工作台属性外未指定任何其它属性，则使用工作台属性

有关启动生成的详细信息，请参阅[在工作台中生成](#)。

相关概念

第 267 页的『[构建描述符部件](#)』

第 7 页的『[开发过程](#)』

第 483 页的『[生成的输出](#)』

相关任务

第 298 页的『[在工作台中生成](#)』

第 108 页的『[设置缺省构建描述符](#)』

相关参考

第 484 页的『[生成的输出（参考）](#)』

从工作台批处理接口生成

要从工作台批处理接口生成，执行下列操作：

1. 确保 Java 类路径提供了对以下 JAR 文件的访问：

- startup.jar，它在以下目录中：

installationDir\eclipse

installationDir

产品安装目录，如 C:\Program Files\IBM\RSPD\6.0。如果在安装您要使用的产品之前安装了 Rational Developer 产品并将它保留了下来，则可能需要指定在先前安装中使用的目录。

- eglutil.jar，它在以下目录中：

installationDir\egl\eclipse\plugins\
com.ibm.etools.egl.utilities_*version*\runtime

installationDir

产品安装目录，如 C:\Program Files\IBM\RSPD\6.0。如果在安装您要使用的产品之前安装了 Rational Developer 产品并将它保留了下来，则可能需要指定在先前安装中使用的目录。

version

插件的已安装版本；例如，6.0.0

2. 确保工作空间中包含生成时所需要的项目和 EGL 部件。

3. 开发 EGL 命令文件。

4. 调用命令 EGLCMD（它可能在生成、运行和测试代码的较大型批处理作业中）。调用 EGLCMD 时指定感兴趣的工作空间。

相关概念

『从工作台批处理接口生成』

相关参考

第 436 页的『EGLCMD』

第 439 页的『EGL 命令文件』

从工作台批处理接口生成

工作台批处理接口是一个功能部件，它允许您从可以访问工作台的批处理环境中生成 EGL 输出。工作台无需正在运行。生成的 EGL 代码只能访问先前已装入到工作空间中的项目和 EGL 部件。

要调用此接口，请使用批处理命令 EGLCMD，该命令同时引用工作空间和 EGL 命令文件。

相关概念

第 7 页的『开发过程』

第 483 页的『生成的输出』

相关任务

『从工作台批处理接口生成』

从 EGL 软件开发包 (SDK) 生成

要从 EGL SDK 生成，执行下列操作：

1. 确保 Java 1.3.1（或更高级别）位于将在其中生成代码的机器上。在安装 EGL 的机器上将自动安装适当级别的 Java 代码。生成机器与目标机器上的 Java 级别必须是兼容的。
2. 确保 `eglbatchgen.jar` 位于 Java 类路径中。JAR 文件在以下目录中：
`installationDir\bin`
`installationDir`
产品安装目录，如 `C:\Program Files\IBM\RSPD\6.0`。如果在安装您要使用的产品之前安装了 Rational Developer 产品并将它保留了下来，则可能需要指定在先前安装中使用的目录。
3. 确保 EGL SDK 能够访问生成时所需要的 EGL 文件
4. （可选）开发 EGL 命令文件
5. 调用命令 `EGLSDK`（它可能在生成、运行和测试代码的较大型批处理作业中）

相关概念

『从 EGL 软件开发包 (SDK) 生成』
第 13 页的『EGL 项目、包和文件』

相关参考

第 435 页的『EGL 构建路径和 `eglp` 路径』
第 436 页的『EGLCMD』
第 439 页的『EGL 命令文件』
第 446 页的『EGLSDK』

从 EGL 软件开发包 (SDK) 生成

EGL 软件开发包 (SDK) 是一个功能部件，它允许您在批处理环境中生成输出，即使您缺少对下列 Rational Developer 产品的访问权亦如此：

- 图形用户界面
- 有关项目组织方式的详细信息

可以使用 EGL SDK 来从软件配置管理 (SCM) 工具（如 Rational ClearCase®）中触发生成，这可能是作为在正常工作时间之后运行的批处理作业的一部分进行的。

要调用 EGL SDK，在批处理文件中或在命令提示符下使用命令 `EGLSDK`。命令调用本身可以采用下列两种格式的其中一种：

- 它可以指定 EGL 文件和构建描述符。在这种情况下，如果您想要有多个生成，则编写多个命令。
- 另外，调用可以引用一个 EGL 命令文件，该文件包含产生一个或多个生成所需的信息。

然而，在组织工作时，可以对 *eglp*（它是在 EGL SDK 使用 `import` 语句来解析部件引用时搜索的一组目录）指定值。另外，必须指定构建描述符选项 **genDirectory** 而不是 **genProject**。

使用 EGLSDK 的先决条件和过程在从 *EGL SDK* 生成中作了描述。有关命令调用的详细信息，请参阅 *EGLSDK*。

相关概念

第 7 页的『开发过程』
第 483 页的『生成的输出』

相关任务

第 302 页的『从 EGL 软件开发包 (SDK) 生成』

相关参考

第 355 页的『genDirectory』
第 436 页的『EGLCMD』
第 435 页的『EGL 构建路径和 *eglp*』
第 446 页的『EGLSDK』

在生成之后调用构建规划

您可能希望创建构建规划并在稍后的时间调用该规划。如果在生成时网络故障阻止您在远程机器上准备代码，就会发生这种情况。

在这种情况下，要调用构建规划，应完成下列步骤：

1. 确保 `eglb` 位于 Java 类路径中，就象在安装 EGL 的机器上会自动发生的那样。JAR 文件在以下目录中：

```
installationDir\egl\ eclipse\plugins\  
com.ibm.etools.egl.batchgeneration_version
```

installationDir

产品安装目录，如 `C:\Program Files\IBM\RSPD\6.0`。如果在安装您要使用的产品之前安装了 Rational Developer 产品并将它保留了下来，则可能需要指定在先前安装中使用的目录。

version

插件的已安装版本；例如，6.0.0

2. 同样，确保 `PATH` 变量包括该目录。
3. 在命令行中输入以下命令：

```
java com.ibm.etools.egl.distributedbuild.BuildPlanLauncher bp
```

bp 构建规划文件的标准路径。有关生成的文件的名称的详细信息，请参阅生成的输出（参考）。

相关概念

第 297 页的『构建规划』
第 293 页的『生成』

相关任务

第 296 页的『构建 EGL 输出』

相关参考

第 347 页的『构建描述符选项』

第 484 页的『生成的输出（参考）』

生成 Java; 其它主题

处理生成到目录中的 Java 代码

本页面描述如何处理生成到目录中的代码。但是，建议您避免将代码生成到目录中；有关详细信息，请参阅将 *Java* 代码生成到项目中。

要将代码生成到目录中，请指定构建描述符选项 **genDirectory**，并避免指定构建描述符选项 **genProject**。

接下来的任务视项目类型而定：

应用程序客户机项目

对于应用程序客户机项目，执行下列操作：

1. 通过将下列条目添加至项目的 Java 构建路径来提供对 EGL jar 文件的准备时访问：

```
EGL_GENERATORS_PLUGINDIR/runtime/fda6.jar  
EGL_GENERATORS_PLUGINDIR/runtime/fdaj6.jar
```

有关每个条目开头部分的变量的详细信息，请参阅设置变量 *EGL_GENERATORS_PLUGINDIR*。

2. 提供对 fda6.jar、fdaj6.jar 和（如果正在通过 TCP/IP 调用生成的程序的话）EGLTcipListener.jar 的运行时访问：

- 从以下目录中访问 jar 文件：

```
installationDir\egl\eclipse\plugins\  
com.ibm.etools.egl.generators_version\runtime
```

installationDir

产品安装目录，如 C:\Program Files\IBM\SPD\6.0。如果在安装您要使用的产品之前安装了 Rational Developer 产品并将它保留了下来，则可能需要指定在先前安装中使用的目录。

version

插件的已安装版本；例如，6.0.0

将那些文件复制到每个引用应用程序客户机项目的企业应用程序项目中。

- 更新应用程序客户机项目中的清单，以使 jar 文件（存储在企业应用程序项目中）可用。
3. 提供对非 EGL jar 文件的访问（可选任务）
 4. 在遵循下列规则的情况下，将生成的输出导入到项目中：

- 文件夹 *appClientModule* 必须包含生成的输出所在的包的顶层文件夹
- *appClientModule* 下方的文件夹名的层次结构必须与 Java 包的名称匹配

例如，如果正从包 *my.trial.package* 导入生成的输出，则必须将该输出导入到位于以下位置中的文件夹中：

```
appClientModule/my/trial/package
```

5. 如果已生成 J2EE 环境文件，则更新该文件
6. 更新部署描述符
7. 既然已经将输出文件放到项目中，您可以继续设置 J2EE 运行时环境了

EJB 项目

对于 EJB 项目，执行下列操作：

1. 通过将下列条目添加至项目的 Java 构建路径来提供对 EGL jar 文件（fda6.jar 和 fdaj6.jar）的准备时访问：

```
EGL_GENERATORS_PLUGINDIR/runtime/fda6.jar
EGL_GENERATORS_PLUGINDIR/runtime/fdaj6.jar
```

有关每个条目开头部分的变量的详细信息，请参阅设置变量 *EGL_GENERATORS_PLUGINDIR*。

2. 提供对 EGL jar 文件的运行时访问：

- 从以下目录中访问 fda6.jar 和 fdaj6.jar：

```
installationDir\egl\eclipse\plugins\
com.ibm.etools.egl.generators_version\runtime
```

installationDir

产品安装目录，如 C:\Program Files\IBM\RPD\6.0。如果在安装您要使用的产品之前安装了 Rational Developer 产品并将它保留了下来，则可能需要指定在先前安装中使用的目录。

version

插件的已安装版本；例如，6.0.0

将那些文件复制到每个引用 EJB 项目的企业应用程序项目中。

- 更新 EJB 项目中的清单，以使 fda6.jar 和 fdaj6.jar（存储在企业应用程序项目中）可用。

3. 提供对非 EGL jar 文件的访问（可选任务）

4. 在遵循下列规则的情况下，将生成的输出导入到项目中：

- 文件夹 *ejbModule* 必须包含生成的输出所在的包的顶层文件夹
- *ejbModule* 下方的文件夹名的层次结构必须与 Java 包的名称匹配

例如，如果正从包 *my.trial.package* 导入生成的输出，则必须将该输出导入到位于以下位置中的文件夹中：

```
ejbModule/my/trial/package
```

5. 如果已生成 J2EE 环境文件，则更新该文件。
6. 更新部署描述符
7. 设置 JNDI 名称
8. 生成部署代码
9. 既然已经将输出文件放到项目中，您可以继续设置 J2EE 运行时环境了

J2EE Web 项目

对于 Web 项目，执行下列操作：

1. 通过将 fda6.jar 和 fdaj6.jar 复制到 Web 项目文件夹中来提供对 EGL jar 文件的访问。为此，导入位于以下目录中的外部 jar：

```
installationDir\egl\eclipse\plugins\
com.ibm.etools.egl.generators_version\runtime
```

installationDir

产品安装目录，如 C:\Program Files\IBM\SPD\6.0。如果在安装您要使用的产品之前安装了 Rational Developer 产品并将它保留了下来，则可能需要指定在先前安装中使用的目录。

version

插件的已安装版本；例如，6.0.0

这些文件的目标位置是以下项目文件夹：

WebContent/WEB-INF/lib

2. 提供对非 EGL jar 文件的访问（可选）
3. 在遵循下列规则的情况下，将生成的输出导入到项目中：

- 文件夹 *WebContent* 必须包含生成的输出所在的包的顶层文件夹
- *WebContent* 下方的文件夹名的层次结构必须与 Java 包的名称匹配

例如，如果正从包 *my.trial.package* 导入生成的输出，则必须将该输出导入到位于以下位置中的文件夹中：

WebContent/my/trial/package

4. 更新部署描述符
5. 既然已经将输出文件放到项目中，您可以继续设置 J2EE 运行时环境了

Java 项目

如果要生成要在非 J2EE 环境中使用的代码，并且使用下列构建描述符选项组合，将生成属性文件：

- **genProperties** 设置为 GLOBAL 或 PROGRAM；并且
- **J2EE** 设置为 NO。

如果请求全局属性文件（**rununit.properties**），EGL 会将该文件放在顶层目录中。如果改为请求程序属性文件，EGL 会将该文件放在程序中，即放在对应包名中的最后一个限定符的文件夹中或顶层目录中。（如果 EGL 源文件中未指定包名，将使用顶层目录。）

在运行时，程序属性文件中的值用来设置标准 JDBC 连接。有关详细信息，请参阅[了解如何建立标准 JDBC 连接](#)。

对于 Java 项目，您的任务如下所示：

1. 通过将下列条目添加至项目的 Java 构建路径来提供对 EGL jar 文件的访问：

```
EGL_GENERATORS_PLUGINDIR/runtime/fda6.jar  
EGL_GENERATORS_PLUGINDIR/runtime/fdaj6.jar
```

有关每个条目开头部分的变量的详细信息，请参阅[设置变量 EGL_GENERATORS_PLUGINDIR](#)。

2. 如果程序访问关系数据库，则确保 Java 构建路径包含驱动程序的安装目录。例如，对于 DB2，指定包含 db2java.zip 的目录。
3. 如果生成的代码访问 MQSeries，则提供对非 EGL jar 文件的访问
4. 确保程序属性文件（如果存在的话）在顶级项目文件夹中，并且全局属性文件（**rununit.properties**）在对应包名中的最后一个限定符的文件夹中或顶级项目文件夹中。（如果 EGL 源文件中未指定包名，将使用顶级文件夹。）
5. 将链接属性文件放在项目中（可选任务）

相关概念

第 293 页的『将 Java 代码生成到项目中』

相关任务

『生成 EJB 项目的部署代码』

第 329 页的『部署链接属性文件』

第 322 页的『设置部署描述符值』

第 330 页的『提供对非 EGL jar 文件的访问』

第 325 页的『设置 EJB 项目的 JNDI 名称』

『设置变量 EGL_GENERATORS_PLUGINDIR』

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

第 240 页的『了解如何建立标准 JDBC 连接』

第 324 页的『手工更新部署描述符』

第 323 页的『更新 J2EE 环境文件』

相关参考

第 355 页的『genDirectory』

第 357 页的『genProject』

生成 EJB 项目的部署代码

在生成到 EJB 项目中并指定部署描述符属性之后，可以生成允许对 EJB 进行远程访问的存根和框架：

1. 在“项目资源管理器”中，右键单击项目名；然后单击**部署**
2. 遵循“从工作台生成 EJB 部署代码”帮助页面上指定的指导

相关任务

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

设置变量 EGL_GENERATORS_PLUGINDIR

工作台类路径变量 EGL_GENERATORS_PLUGINDIR 包含工作台中的 EGL 插件的标准路径。在将 EGL 程序生成到 Java、应用程序客户机或 EJB 类型的项目中时，将在 Java 构建路径中使用此变量。

如果您遇到涉及 EGL_GENERATORS_PLUGINDIR 的类路径错误，则可能是未设置此变量。例如，如果从软件配置管理系统（如并发版本控制系统（CVS））中检出与 EGL 相关的项目，但您从来没有使用过 EGL 部件，则会发生此问题。

您可以通过创建 EGL 部件、通过生成 EGL 代码或通过执行下列步骤来设置此变量：

1. 选择**窗口**，然后选择**首选项**
2. 在“首选项”页上，选择 **Java**，然后选择**类路径变量**
3. 选择**新建...**
4. 在“新建变量条目”页上，输入 **EGL_GENERATORS_PLUGINDIR**，并指定以下目录：

```
installationDir\egl\eclipse\plugins\  
com.ibm.etools.egl.generators_version
```

installationDir

产品安装目录，如 C:\Program Files\IBM\RSPD\6.0。如果在安装您要使用的产品之前安装了 Rational Developer 产品并将它保留了下来，则可能需要指定在先前安装中使用的目录。

version

插件的已安装版本；例如，6.0.0

在设置此变量之后，请重建项目。

相关概念

第 293 页的『将 Java 代码生成到项目中』

相关参考

第 357 页的『genProject』

在本地机器上运行 EGL 生成的 Java 代码

在本地机器上启动基本或文本用户界面 Java 应用程序

要在本地机器上启动 EGL 生成的基本（批处理）或文本用户界面（TUI）Java 应用程序，执行下列操作：

1. 从 EGL 源代码生成 Java 源代码；有关详细信息，请参阅在工作台中生成。
2. 在“项目资源管理器”中，展开 **JavaSource** 文件夹并选择想要运行的应用程序的 Java 源文件。
3. 在“工作台”菜单上，选择 **运行 > 运行方式 > Java 应用程序**；或在“工作台”工具栏上，单击 **运行按钮** 旁边的向下箭头，然后选择 **运行方式 > Java 应用程序**。

相关概念

第 300 页的『工作台中的生成』

相关任务

第 298 页的『在工作台中生成』

在本地机器上启动 Web 应用程序

如果要使用基于 EGL 的 Web 应用程序来访问 JNDI 数据源，则只有先前配置了 Web 应用程序服务器时，才能遵循当前主题中的指示信息。有关特定于 WebSphere 的后台信息，请参阅 *WebSphere Application Server* 和 *EGL*。

要启动 Web 应用程序，遵循下列步骤：

1. 从 EGL 源代码生成 Java 源代码；有关详细信息，请参阅在工作台中生成。
2. 在“项目资源管理器”中，展开 **WebContent** 和 **WEB-INF** 文件夹。右键单击想要运行的 JSP，然后选择 **运行 > 在服务器上运行**。将显示“选择服务器”对话框。
3. 如果已经为此 Web 项目配置了服务器，选择 **选择现有服务器**，然后从列表中选择服务器。单击 **完成** 以启动服务器，将应用程序部署至服务器并启动该应用程序。
4. 如果尚未为此 Web 项目配置服务器，可按如下所示进行处理，但仅当应用程序未访问 JNDI 数据源时才能这样做：
 - a. 选择 **手工定义服务器**。

- b. 指定主机名，即（对于本地机器）**localhost**。
- c. 选择一种服务器类型，该类型与您打算在运行时在其上部署应用程序的 Web 应用程序服务器类似。选择包括 **WebSphere V5.1 测试环境**和 **WebSphere V6.0 服务器**。
- d. 如果在处理当前项目时不打算更改您的选择，则选择**将服务器设置为缺省项目服务器**复选框。
- e. 在大多数情况下，可避免此步骤；但如果您希望指定不同于缺省值的设置，则单击**下一步**并进行选择。
- f. 单击**完成**以启动服务器，将应用程序部署至服务器并启动该应用程序。

相关概念

第 300 页的『工作台中的生成』

『WebSphere Application Server 和 EGL』

第 171 页的『Web 支持』

相关任务

第 298 页的『在工作台中生成』

WebSphere Application Server 和 EGL

在工作台中运行或调试用 EGL 编写的 J2EE 应用程序时，可能会使用下列其中一个 IBM 运行时环境：

- WebSphere V5.1 测试环境，它支持 Java servlet V2.3（及之前版本）和 EJB V2.0（及之前版本）
- WebSphere Application Server V6.0，它支持 Java servlet V2.4（及之前版本）和 EJB V2.1（及之前版本）

如果要调试或运行未使用 J2EE 数据源的代码，则在两个环境中运行代码的过程是类似的，只需要单击几下鼠标。

但是，如果需要访问 J2EE 数据源，则情况如下所示：

- 如果要使用 WebSphere V5.1 测试环境，则以任意顺序执行下列两个步骤：
 1. 在定义服务器配置时标识数据源。
 2. 确保应用程序引用的是该数据源的服务器配置条目。

第二个步骤包括在特定于项目的部署描述符中指定 JNDI 名称。以下列任一方式指定 JNDI 名称：

- 在创建项目时；或者
- 在更新部署描述符时。

有关服务器配置的详细信息，请参阅配置 *WebSphere Application Server V5.x*。

- 如果要使用 WebSphere Application Server V6.0，则以任意顺序执行下列两个步骤：
 1. 可以以下列任一方式对服务器标识数据源：
 - 更新应用程序部署描述符（`application.xml`）时，这是建议的方式；或者
 - 在“管理控制台”上配置服务器时。

有关更新应用程序部署描述符的详细信息，请参阅[设置服务器以便测试 WebSphere Application Server V6.0 的数据源](#)。有关使用“管理控制台”的详细信息，请参阅[配置 WebSphere Application Server V6.x](#)。

2. 确保应用程序引用的是该数据源的服务器配置条目。

第二个步骤包括在特定于项目的部署描述符中指定 JNDI 名称。以下列任一方式指定 JNDI 名称:

- 在创建项目时; 或者
- 在更新部署描述符时。

更新应用程序部署描述符比使用“管理控制台”有以下优点:

- 可将企业应用程序部署至支持 J2EE V1.4 的任何 Web 应用程序服务器，在标识数据源时不需要额外的服务器配置。
- 不管服务器是否在运行，都可以更新应用程序部署描述符。
- 因为操作都是在 Rational Developer 产品的开发组件（而不是在 WebSphere Application Server 组件）中进行的，所以方便许多。

不管您怎样更新数据源信息，更改都会立即体现在服务器上。

相关概念

第 171 页的『Web 支持』

构建脚本

构建脚本是一个由构建规划调用的文件，它根据生成的文件而准备输出。示例如下所示:

- 为开发系统上的构建服务器提供了 Java 编译器或其它 .exe（二进制）文件或 .bat（文本）文件，此文件发送至远程 Windows 2000/NT/XP 上的构建服务器。
- 脚本（.scr 文件）或一些二进制代码发送至 USS 构建服务器。

通过设置构建描述符选项 **destHost** 来指定构建机器的地址。

Java 构建脚本

为了准备 Java 代码以便执行它们，EGL 将 javac（Java 编译器）命令及其参数放在构建规划中，并将 javac 命令和该命令所需的输入发送至构建机器。

相关概念

第 295 页的『构建』

第 297 页的『构建规划』

第 311 页的『构建服务器』

相关参考

第 347 页的『构建描述符选项』

第 352 页的『destDirectory』

第 353 页的『destHost』

第 353 页的『destPassword』

第 354 页的『destUserID』

第 614 页的『Java 程序生成输出』
第 615 页的『Java 包装器生成输出』

构建服务器

构建服务器接收来自客户机系统的以下请求：根据从该客户机发送的源代码创建可执行文件。必须在从构建客户机发送任何请求之前启动构建服务器。构建服务器通常为来自多个客户机的请求提供服务。如果接收到并发构建请求，则可启动多个线程。

在生成器环境中，您在操作系统是目标生成系统（例如，Windows 2000）的机器上启动构建服务器。生成器将生成 Java 源代码。Java 代码发送至在其中调用了 Java 编译器的指定构建服务器。

如果正在为 Windows 生成 Java 代码，则可以在执行了生成的机器上构建 Java 输出。这称为本地构建。在这种情况下，不必启动构建服务器。如果要执行本地构建，则省略构建描述符的 **destHost** 选项。

相关概念

第 295 页的『构建』
第 310 页的『构建脚本』

相关任务

『在 AIX、Linux 或 Windows 2000/NT/XP 上启动构建服务器』

相关参考

第 347 页的『构建描述符选项』

在 AIX、Linux 或 Windows 2000/NT/XP 上启动构建服务器

要在 AIX、Linux 或 Windows 2000/NT/XP 上启动远程构建服务器，请在“命令提示符”窗口中输入 `ccublds` 命令。语法如下所示：

```
►► ccublds -p portno -v -a 0/2
```

其中

-p 指定服务器侦听的端口号 (*portno*)，服务器使用此端口号来与客户机通信。

-v 指定服务器的冗余级别。最多可以指定此参数三次（最大冗余）。

-a 指定认证方式：

0 服务器将执行由任何客户机请求的构建。仅在对安全性要求不高的环境中才建议使用此方式。

2 在接受构建之前，服务器将要求客户机提供有效的用户标识和密码。用户标识和密码最初是由运行构建服务器的主机的所有者配置的。通过使用下面所描述的“安全管理器”来执行配置。

设置从构建服务器返回的消息的语言

Windows 上的构建服务器以下表中列示的任何语言返回消息，缺省语言是英语。

语言	代码
巴西葡萄牙语	ptb
简体中文	chs
繁体中文	cht
美国英语	enu
法语	fra
德语	deu
意大利语	ita
日语	jpn
韩国语	kor
西班牙语	esp

要指定除了英语之外的语言，应确保在启动构建服务器之前将环境变量 `CCU_CATALOG` 设置为非英语消息目录。需要的值格式如下（在一行上）：

```
installationDir\egl\ eclipse\plugins
\com.ibm.etools.egl.distributedbuild\executables
\ccu.cat.xxx
```

installationDir

产品安装目录，如 `C:\Program Files\IBM\RSPD\6.0`。如果在安装您要使用的产品之前安装了 Rational Developer 产品并将它保留了下来，则可能需要指定在先前安装中使用的目录。

xxx

构建服务器支持的语言代码；上表中列示的其中一个代码

安全管理器

“安全管理器”是一个服务器程序，构建服务器使用该程序来验证发送构建请求的客户端。

为“安全管理器”设置环境： “安全管理器”使用下列 Windows 环境变量：

CCUSEC_PORT

将端口号设置为“安全管理器”侦听的端口号。缺省值为 22825。

CCUSEC_CONFIG

设置将配置数据保存在的文件的路径名。缺省文件为 `C:\temp\ccuconfig.bin`。如果找不到此文件，则“安全管理器”将创建该文件。

CCU_TRACE

如果将此变量设置为 `*`，则将启动跟踪“安全管理器”，以便于进行诊断。

启动“安全管理器”： 要启动“安全管理器”，发出以下命令：

```
java com.ibm.etools.egl.distributedbuild.security.CcuSecManager
```

配置“安全管理器”： 要配置“安全管理器”，使用“配置工具”，它具有图形界面。可以通过发出以下命令来运行该工具：

```
java com.ibm.etools.egl.distributedbuild.security.CCUconfig
```

当“配置工具”正在运行时，选择**服务器项**选项卡。使用按钮“添加...”，要添加想要构建服务器支持的用户，单击**添加...**按钮。必须为用户标识定义密码。可以为用户定义下列限制和特权：

- 此用户可以指定的位置，即，`ccubldc` 命令的 `-la` 参数的值。不同的位置之间用分号隔开。
- 此用户可以指定的构建脚本的名称。（EGL 构建规划仅将 `javac` 命令用作构建脚本。）
- 此用户是否可以从客户机发送构建脚本，即，使用 `ccubldc` 命令的 `-ft` 参数。（EGL 生成器不使用 `-ft` 参数。如果正将构建用于除准备 Java 生成输出以外的其它用途，则应指定此参数。）

这些定义保存在持久存储器中，位于由 `CCUSEC_CONFIG` 指定的一个文件中，并且在各个会话之间会记住这些定义。

相关概念

第 310 页的『构建脚本』

第 311 页的『构建服务器』

相关任务

第 690 页的『EGL 语句和命令的语法图』

部署 EGL 生成的 Java 输出

Java 运行时属性

EGL 生成的 Java 程序使用一组运行时属性，这些属性提供诸如如何访问由程序使用的数据库和文件之类的信息。

在 J2EE 环境中

对于将在 J2EE 环境中运行的生成的 Java 程序，下列情况是有可能的：

- EGL 可以将运行时属性直接生成到 J2EE 部署描述符中。在这种情况下，EGL 将覆盖已存在的属性并追加不存在的属性。在运行时，程序将访问 J2EE 部署描述符。
- 另外，EGL 可以将运行时属性生成到 J2EE 环境文件中。可以定制该文件中的属性，然后将它们复制到 J2EE 部署描述符中。
- 完全可以不生成运行时属性，在这种情况下，您必须手工编写任何需要的属性。

在 J2EE 模块中，因为模块中的所有代码都共享同一部署描述符，所以每个程序都具有相同的运行时属性。

在 WebSphere Application Server 中，属性在与 Web 项目相关联的 web.xml 文件中被指定为 env-entry 标记，如以下示例中所示：

```
<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>
```

在非 J2EE Java 环境中

对于在 J2EE 环境外部运行的生成的 Java 程序，可以将运行时属性生成到程序属性文件中，也可以手工编写该文件。（程序属性文件提供在部署描述符中可用的信息类型，但是属性的格式不同。）

在非 J2EE Java 环境中，可以在其中任何属性文件中指定属性，它们是按以下顺序进行搜索的：

- **user.properties**
- 具有以下名称的文件：
 - programName.properties*
 - programName*
运行单元中的第一个程序
- **rununit.properties**

在指定特定于用户的属性时适合使用 **user.properties**。EGL 不生成此文件的内容。

当运行单元的第一个程序不访问文件或数据库，而是调用执行那些操作的程序时，就特别适合于使用 **rununit.properties**：

- 当生成调用程序时，可以生成具有该程序的名称的属性文件，内容可以不包含与数据库或文件相关的属性
- 当生成被调用程序时，可以生成 **rununit.properties**，内容将可供这两个程序使用

这些文件都不是必需的，简单的程序不需要它们中的任何一个。

在部署时，下列规则适用：

- 用户属性文件（**user.properties**，如果有的话）在用户主目录中，这是由 Java 系统属性 *user.home* 确定的。
- 程序属性文件（如果存在的话）的位置取决于该程序是否在包中。下面的示例很好地说明了这些规则：
 - 如果程序 P 在包 x.y.z 中并且部署至 MyProject/JavaSource，则程序属性文件必须在 MyProject/JavaSource/x/y/z 中
 - 如果程序 P 不在包中并且部署至 myProject/JavaSource，则程序属性文件（如全局属性文件）必须在 MyProject/JavaSource 中

在任何一种情况下，MyProject/JavaSource 都必须在该类路径中。

- 全局属性文件（**rununit.properties**，如果存在的话）必须与程序在一起，位于在类路径中指定的目录中。

如果生成 Java 项目的输出，EGL 会将属性文件（并非 **user.properties**）放在适当的文件夹中。

如果要生成 Java 代码以便在与使用较早版本的 EGL 或 VisualAge Generator 生成 Java 代码的相同的运行单元中使用，用于部署属性文件的规则取决于运行单元中的第一个程序是否是使用 EGL 6.0 或更新版本生成的（在这种情况下，此处描述的规则适用），或者是使用较早版本的 EGL 或 VisualAge Generator 生成的（在这种情况下，属性文件可能在类路径中的任何目录中，并且全局文件被称为 **vgj.properties**）。

最后，如果第一个程序是使用较早版本的软件生成的，可指定另一个属性文件，该文件在运行单元中全局使用以代替所有非全局属性文件。有关详细信息，请参阅 *Java 运行时属性（详细信息）* 中的属性 **vgj.properties.file** 的描述。

构建描述符和程序属性

选项作为构建描述符选项值被提交至 EGL：

- 要将属性生成到 J2EE 部署描述符中，请将 **J2EE** 设置为 YES，将 **genProperties** 设置为 PROGRAM 或 GLOBAL，并生成到 J2EE 项目中。
- 要将属性生成到 J2EE 环境文件中，请将 **J2EE** 设置为 YES，将 **genProperties** 设置为 PROGRAM 或 GLOBAL，并执行下列其中一项操作：
 - 生成到目录中（在这种情况下，使用构建描述符选项 **genDirectory**，而不是使用 **genProject**）；或者
 - 生成到非 J2EE 项目中。
- 要生成与正在生成的程序同名的程序属性文件，请将 **J2EE** 设置为 NO，将 **genProperties** 设置为 PROGRAM，并生成到除 J2EE 项目以外的项目中。

- 要生成程序属性文件 **rununit.properties**，将 **J2EE** 设置为 NO；将 **genProperties** 设置为 GLOBAL；并将其生成到 J2EE 项目之外的项目中。
- 要避免生成属性，将 **genProperties** 设置为 NO。

有关其它信息

有关将属性生成到部署描述符中或 J2EE 环境文件中的详细信息，请参阅 [设置部署描述符值](#)。

有关运行时属性的含义的详细信息，请参阅 [Java 运行时属性（详细信息）](#)。

有关访问 EGL 代码中的运行时属性的详细信息，请参阅 [sysLib.getProperty](#)。

相关概念

第 255 页的『EGL 调试器』

第 293 页的『将 Java 代码生成到项目中』

第 324 页的『J2EE 环境文件』

『程序属性文件』

第 678 页的『运行单元』

相关任务

第 304 页的『处理生成到目录中的 Java 代码』

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

第 322 页的『设置部署描述符值』

第 324 页的『手工更新部署描述符』

第 323 页的『更新 J2EE 环境文件』

相关参考

第 358 页的『genProperties』

第 360 页的『J2EE』

第 493 页的『Java 运行时属性（详细信息）』

第 828 页的『getProperty()』

为 EGL 生成的代码设置非 J2EE 运行时环境

程序属性文件

程序属性文件包含 Java 运行时属性，这些运行时属性具有特殊的格式，只能由运行于 J2EE 环境外部的 Java 程序访问。有关概述信息，请参阅 [Java 运行时属性](#)。

程序属性文件是文本文件。除了注释之外的每个条目都具有以下格式：

```
propertyName = propertyValue
```

propertyName

在 [Java 运行时属性（详细信息）](#) 中描述的其中一项属性

propertyValue

在运行时可供程序使用的属性值

注释是任何这样的行，该行中的第一个非文本字符是磅符（#）。

示例文件的一部分如下所示：

```
# This file contains properties for generated
# Java programs that are being debugged in a
# non-J2EE Java project
```

```
vgj.nls.code = ENU
vgj.datemask.gregorian.long.ENU = MM/dd/yyyy
```

有关对生成的文件指定的名称的详细信息，请参阅生成的输出（参考）。

相关概念

第 255 页的『EGL 调试器』

第 315 页的『Java 运行时属性』

相关任务

第 484 页的『生成的输出（参考）』

相关参考

第 358 页的『genProperties』

第 360 页的『J2EE』

第 493 页的『Java 运行时属性（详细信息）』

在 J2EE 外部部署 Java 应用程序

要在 J2EE 外部部署 Java 应用程序，执行下列操作：

1. 遵循安装用于 Java 的 EGL 运行时代码中描述的过程
2. 将 EGL 生成的代码导出至 JAR 文件，记住包括生成的输出文件（具有 java 之外的扩展名）；如 jasper、属性和 tab 文件
3. 将任何手工编写的 Java 代码导出到 JAR 文件中
4. 将导出的文件包括在目标机器的类路径中

相关任务

『安装用于 Java 的 EGL 运行时代码』

安装用于 Java 的 EGL 运行时代码

以下 Web 站点以 zip 文件的形式为生成的 Java 应用程序提供 EGL 运行时代码：

<http://www3.software.ibm.com/ibmdl/pub/software/rationalsdp/rad/60/redist>

受支持的分布式平台包括：AIX、HP-UX、Linux（Intel™）、iSeries、Solaris 和 Windows 2000/NT/XP。（请参阅产品先决条件以了解受支持的版本。）EGL 为 AIX、HP-UX 和 Solaris 提供 32 位和 64 位支持。

从先前提到的 Web 站点下载的 zip 文件包括下列内容：

- JAR 文件，包含所有受支持分布式平台共用的 Java 代码
- 特定于平台的代码

执行下列操作：

1. 将 EGLRuntimes 目录中的文件解压缩至每台机器，在每台机器上，已部署的 EGL 应用程序将在 J2EE 应用程序服务器外部运行。（这些文件已经包含在任何用来部署 J2EE 应用程序的企业归档（EAR）文件中。）
2. 将这些 JAR 文件包括在部署机器的类路径中。

3. 将所有特定于平台的代码复制至每台部署机器上的某个目录；然后为每台机器设置相应环境变量：

对于 AIX (32 位或 64 位支持)

有意义的文件将放在目录 **Aix** 或 (对于 64 位支持) **Aix64** 中。更改 **PATH** 和 **LIBPATH** 环境变量以便让它们引用包含您从 Web 站点复制的特定于平台的代码的目录。

对于 HP-UX (32 或 64 位支持)

有意义的文件将放在目录 **hpux** 或 (对于 64 位支持) **hpux64** 中。更改 **PATH** 和 **LIBPATH** 环境变量以便让它们引用包含您从 Web 站点复制的特定于平台的代码的目录。

对于 iSeries

有意义的文件将放在目录 **Iseries** 中。在 **qshell** 中，切换到刚刚将文件上传到其中的目录，然后运行 **setup.sh** 脚本并指定“install”选项：

```
> setup.sh install
```

另外，必须设置其它一些环境变量。有关如何设置这些环境变量的信息，请运行该脚本并指定“envinfo”选项：

```
> setup.sh envinfo
```

如果由于某些原因而删除了在安装期间为您创建的 **symlink**，则可以通过“link”选项来重新创建它：

```
> setup.sh link
```

对于 Linux

有意义的文件将放在目录 **Linux** 中。更改 **PATH** 和 **LIBPATH** 环境变量以便让它们引用包含您从 Web 站点复制的特定于平台的代码的目录。

对于 Solaris (32 或 64 位支持)

有意义的文件将放在目录 **Solaris** 或 (对于 64 位支持) **Solaris64** 中。更改 **PATH** 和 **LIBPATH** 环境变量以便让它们引用包含您从 Web 站点复制的特定于平台的代码的目录。

对于 Windows 2000/NT/XP

有意义的文件将放在目录 **Win32** 中。更改 **PATH** 环境变量以便让它引用包含您从 Web 站点复制的特定于平台的代码的目录。

相关任务

第 318 页的『在 J2EE 外部部署 Java 应用程序』

将 JAR 文件包括在目标机器的 CLASSPATH 中

可将包含 EGL 生成的代码的所有 JAR 文件或手工编写的 Java 代码包括在目标机器的 **CLASSPATH** 中。此程序的步骤与系统有关。有关详细信息，请参阅您的操作系统文档。

为 EGL 运行时设置 UNIX curses 库

在 AIX 或 Linux 上部署 EGL 文本程序时，EGL 运行时会尝试使用 **UNIX curses** 库。如果该环境不是为 **UNIX curses** 库设置的或者如果该库不受支持，则 EGL 运行时将尝试使用 Java Swing 技术；如果连该技术也未提供的话，程序会失败。

当用户从终端仿真器窗口或字符终端运行 EGL 程序时，UNIX curses 库是必需的。

为使 EGL 运行时能够访问 AIX 或 Linux 上的 UNIX curses 终端库，必须完成 UNIX shell 环境中的几个步骤。在前两个步骤的每个步骤中，*installDir* 都是指运行时安装库：

1. 修改 LD_LIBRARY_PATH 环境变量以包括共享对象 libCursesCanvas6.so，该对象是在运行时安装库中提供的：

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH: /installDir/bin
```

2. 修改 CLASSPATH 环境变量以添加 fda6.jar 和 fdaj6.jar：

```
export CLASSPATH=$CLASSPATH:  
/installDir/lib/fda6.jar: /installDir/lib/fdaj6.jar
```

前面的信息必须在单独的一行上键入。

3. 将 TERM 环境变量设置为适当的终端设置，如以下示例所示：

```
export TERM=vt100
```

如果发生终端异常，则尝试不同的终端设置，诸如 xterm、dtterm 或 vt220。

4. 从 UNIX shell 中运行 EGL Java 程序，如以下示例所示：

```
java myProgram
```

确保 CLASSPATH 环境变量标识程序所在的目录。

有关使用 UNIX 上的 Curses 库的其它详细信息，请参阅 UNIX 联机帮助页。

为调用的非 J2EE 应用程序设置 TCP/IP 侦听器

如果要想让调用程序使用 TCP/IP 来与被调用的非 J2EE Java 程序交换数据，则必须为被调用程序设置 TCP/IP 侦听器。

如果正在使用 TCP/IP 来与被调用的非 J2EE Java 程序通信，则必须为该程序配置名为 CSOTcpipListener 的独立 Java 程序。明确地说，必须执行下列操作：

- 确保运行 CSOTcpipListener 时使用的类路径包含 fda6.jar、fdaj6.jar 以及包含被调用程序的目录或归档；并且
- 将 Java 运行时属性 **tcpiplistener.port** 设置为 CSOTcpipListener 接收数据时使用的端口号。

可以按照下列两种方法中的任何一种方法来启动独立 TCP/IP 侦听器：

- 要从工作台启动侦听器，使用 Java 应用程序的启动配置。在这种情况下，可以在启动配置的程序自变量中指定属性文件的名称。另外，如果正在使用缺省的 `tcpiplistener.properties` 文件，那么该文件不应该位于某个文件夹中，而是应该直接位于您创建启动配置时指定的项目之下。
- 要从命令行启动侦听器，按照如下方式运行程序：

```
java CSOTcpipListener propertiesFile  
propertiesFile
```

TCP/IP 侦听器使用的属性文件的标准路径。如果未指定属性文件，则侦听器将尝试打开当前目录中的以下文件：

```
tcpiplistener.properties
```

相关任务

第 330 页的『提供对非 EGL jar 文件的访问』

为 EGL 生成的代码设置 J2EE 运行时环境

EGL 生成的 Java 程序和包装器在运行时配置中列示的平台上的 J2EE 1.4 服务器（如 WebSphere Application Server V6.0）上运行。

在将生成的 Java 类嵌入 J2EE 模块时，主要任务如下所示：

1. 通过下列两种方法中的任何一种方法将输出文件放到项目中：
 - 生成到项目中，这是首选技术；或者
 - 生成到目录中，然后将文件导入到项目中。
2. 将链接属性文件放在模块中（请参阅部署链接属性文件）。
3. 消除重复的 jar 文件。
4. 导出企业归档（.ear）文件，此文件可能包含 Web 应用程序归档（.war）文件和其它 .ear 文件；有关此过程的详细信息，请参阅有关导出的帮助页面。
5. 将 .ear 文件导入到将要主管应用程序的 J2EE 服务器中；有关此过程的详细信息，请参阅 J2EE 服务器文档。

可能还需要完成下列任务：

- 第 328 页的『设置 J2EE JDBC 连接』
- 第 325 页的『为 CICSJ2C 调用设置 J2EE 服务器』
- 第 326 页的『为 J2EE 应用程序客户机模块中的被调程序设置 TCP/IP 侦听器』
- 第 320 页的『为调用的非 J2EE 应用程序设置 TCP/IP 侦听器』

相关概念

第 7 页的『开发过程』

第 293 页的『将 Java 代码生成到项目中』

第 297 页的『Java 程序、PageHandler 和库』

第 282 页的『链接选项部件』

第 330 页的『链接属性文件』

第 8 页的『运行时配置』

相关任务

第 318 页的『在 J2EE 外部部署 Java 应用程序』

第 329 页的『部署链接属性文件』

第 322 页的『消除重复的 jar 文件』

第 307 页的『生成 EJB 项目的部署代码』

第 304 页的『处理生成到目录中的 Java 代码』

第 330 页的『提供对非 EGL jar 文件的访问』

第 322 页的『设置部署描述符值』

第 325 页的『设置 EJB 项目的 JNDI 名称』

第 328 页的『设置 J2EE JDBC 连接』

第 325 页的『为 CICSJ2C 调用设置 J2EE 服务器』

第 326 页的『为 J2EE 应用程序客户机模块中的被调程序设置 TCP/IP 侦听器』

第 240 页的『了解如何建立标准 JDBC 连接』

第 324 页的『手工更新部署描述符』

第 323 页的『更新 J2EE 环境文件』

相关参考

第 493 页的『Java 运行时属性（详细信息）』

第 598 页的『链接属性文件（详细信息）』

消除重复的 jar 文件

如果将多个 J2EE 模块放在单个 ear 文件中，则执行下列操作来消除重复的 jar 文件：

1. 将每个重复的 jar 文件的副本移至 ear 的顶层
2. 从 J2EE 模块中删除重复的 jar 文件
3. 确保每个受影响的 J2EE 模块的构建路径指向 ear 中的 jar 文件；明确地说，对于那些受影响的每个 J2EE 模块执行以下操作：
 - a. 在“项目资源管理器”或 J2EE 视图中，右键单击模块
 - b. 选择**编辑模块依赖项**
 - c. 当“模块依赖项”对话框显示时，从 ear 的顶层选择要访问的 jar 文件，然后单击**完成**。

相关任务

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

设置部署描述符值

一项重要的任务是将运行时值（类似于环境变量值）放到 J2EE 模块的部署描述符中。例如，可以与下表中列示的工作台编辑器进行交互；并且，在任何情况下，如果您希望重新指定值，都可以使用编辑器来重新指定值。

项目类型	部署描述符的名称	如何指定值
应用程序客户机	application-client.xml	使用 XML 编辑器，“设计”选项卡
EJB	ejb-jar.xml	使用 EJB 编辑器，Bean 选项卡
J2EE Web	web.xml	使用 web.xml 编辑器，“环境”选项卡

如果符合下列所有条件，就会自动添加内容，这是更新部署描述符的建议方法：

- 正在生成 Java 程序或包装器
- 构建描述符选项 **genProperties** 设置为 GLOBAL 或 PROGRAM
- 正在为 J2EE 运行时生成（通过将 **J2EE** 设置为 YES）
- 将 **genProject** 设置为有效的 J2EE 项目

EGL 不会从现有部署描述符中删除属性，而是执行下列操作：

- 覆盖已存在的属性
- 追加不存在的属性

更新部署描述符的另一种方法是从 J2EE 环境文件粘贴值，该文件是在下列所有条件都符合时生成的输出：

- 正在生成 Java 程序
- 构建描述符选项 **genProperties** 设置为 GLOBAL 或 PROGRAM
- 正在为 J2EE 运行时生成（通过将 **J2EE** 设置为 YES）
- 未将 **genProject** 设置为有效的 J2EE 项目，而是如同生成到目录中那样

在从 J2EE 环境文件将条目粘贴到应用程序客户机或 EJB 项目的部署描述符时，需要更改文件中条目的顺序，如更新 J2EE 环境文件中所述。如果正在使用 J2EE Web 项目，则不需要更改条目的顺序。

有关部署描述符属性的详细信息，请参阅 *Java 运行时属性（详细信息）*。

相关概念

- 第 324 页的『J2EE 环境文件』
- 第 293 页的『将 Java 代码生成到项目中』
- 第 317 页的『程序属性文件』

相关任务

- 第 304 页的『处理生成到目录中的 Java 代码』
- 第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』
- 『更新 J2EE 环境文件』
- 第 324 页的『手工更新部署描述符』

相关参考

- 第 355 页的『genDirectory』
- 第 358 页的『genProperties』
- 第 360 页的『J2EE』
- 第 493 页的『Java 运行时属性（详细信息）』

更新 J2EE 环境文件

J2EE 环境文件包含一系列条目，如以下示例所示：

```
<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>
```

子元素的顺序是名称、值和类型。对于 J2EE Web 项目，这是正确的；但是，对于应用程序客户机和 EJB 项目，需要将顺序更改为名称、类型和值。对于上面的示例，将子元素的顺序更改为：

```
<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>
```

如果直接生成到项目中而不是生成到目录中，则不需要执行此步骤。当生成到项目中时，EGL 可以确定正在使用的项目类型并以适当的顺序生成环境条目。

相关任务

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

第 322 页的『设置部署描述符值』

相关参考

第 493 页的『Java 运行时属性（详细信息）』

J2EE 环境文件

J2EE 环境文件是一个文本文件，它包含根据您在生成 Java 程序时指定的信息而派生的属性 - 值对。信息源是构建描述符、资源关联部件和链接选项部件。

当配置 Java 程序的环境时，可以将 *J2EE* 环境文件作为放置在运行时部署描述符中的信息的基础。

有关 *J2EE* 环境文件名称的详细信息，请参阅生成的输出（参考）。

有关设置部署描述符值的不同方法的详细信息，请参阅设置部署描述符值。

相关概念

第 8 页的『运行时配置』

相关任务

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

第 322 页的『设置部署描述符值』

相关参考

第 484 页的『生成的输出（参考）』

第 358 页的『genProperties』

第 364 页的『sqlDB』

手工更新部署描述符

如果正在从生成的 *J2EE* 环境文件中更新部署描述符，则执行下列操作：

1. 阅读设置部署描述符值中的概述信息。
2. 如果已开发应用程序客户机或 EJB 项目，则必须确保生成的环境条目中的子元素的顺序正确，如更新 *J2EE* 环境文件中所述。
3. 将环境条目复制到项目的部署描述符中，如下所示：
 - a. 创建部署描述符的备份副本。
 - b. 打开 *J2EE* 环境文件，该文件称为 *programName-env.txt* 文件。将环境条目复制到剪贴板中。
 - c. 双击部署描述符。
 - d. 单击“源代码”选项卡。
 - e. 在正确的位置粘贴条目。

有关部署描述符的详细信息，请参阅 *Java 运行时属性（详细信息）*。

相关任务

第 322 页的『设置部署描述符值』

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

第 323 页的『更新 J2EE 环境文件』

相关参考

第 493 页的『Java 运行时属性（详细信息）』

设置 EJB 项目的 JNDI 名称

要设置 EJB 项目的 JNDI 名称，执行下列操作：

1. 右键单击 `ejb-jar.xml`（部署描述符）以打开上下文菜单。
2. 使用“EJB 编辑器”来打开项目中的以下文件：
`\ejbModule\META-INF\ejb-jar.xml`
3. 单击 **Bean** 选项卡。
4. 在列表中，单击刚刚生成的 EJB 的名称。
5. 在“WebSphere 绑定”下面输入 JNDI 名称。JNDI 名称必须具有以下特征才能供 EGL 运行时代码使用：
 - 程序名的第一个字符是大写的
 - 程序名的后续字符是小写的
 - 字母 **EJB** 是大写的。

相关任务

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

为 CICSJ2C 调用设置 J2EE 服务器

必须在 J2EE 服务器中为通过协议 CICSJ2C 访问的每个 CICS 事务设置 `ConnectionFactory`。

如果生成的 Java 包装器正在进行 CICSJ2C 调用，则可以按照下列任何方式来处理安全性（其中，包装器指定的值覆盖 J2EE 服务器的值）：

- 在包装器的 `CSOCallOptions` 对象中设置用户标识和密码；或者
- 在 J2EE 服务器中的 `ConnectionFactory` 配置中设置用户标识和密码；或者
- 设置 CICS 区域，以便不需要进行用户认证。

当从 WebSphere 390 中调用程序时，下列限制适用：

- 如果将 `callLink` 元素属性 **luwControl** 设置为 `CLIENT`，则调用会失败。WebSphere 390 连接实现不支持扩展工作单元。
- 部署描述符属性 **cso.cicsj2c.timeout** 的设置不起作用。

缺省情况下，不会发生超时。然而，在宏 `DFHXCOPT` 生成的 `EXCI` 选项表中，可以设置参数 `TIMEOUT`，该参数允许您指定 `EXCI` 等待 `DPL` 命令（`ECI` 请求）完成的时间。设置为 0 意味着无限期等待。

有关详细信息，请参阅 *Java Connectors for CICS: Featuring the J2EE Connector Architecture* (SG24-6401-00)，可从 web 站点 <http://www.redbooks.ibm.com> 获得该出版物。

相关任务

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

为 J2EE 应用程序客户机模块中的被调用程序设置 TCP/IP 侦听器

如果要想让调用程序使用 TCP/IP 来与 J2EE 应用程序客户机模块中的被调用程序交换数据，则必须为被调用程序设置 TCP/IP 侦听器。

您需要确保下列情况属实：

- 特定于 EGL 的 TCP/IP 侦听器是模块的 main 类，这是在模块的清单 (.MF) 文件中指定的
- 对侦听器指定了端口，这是在模块的部署描述符 (application-client.xml) 中指定的

如果正在 J2EE 1.2 级别上使用项目，则建议您先建立随侦听器一起初始化的应用程序客户机项目，然后再将任何 EGL 代码生成到该项目中。如果未遵循该顺序（首先是侦听器，其次才是 EGL 代码），或者如果正在 J2EE 1.3 级别上使用项目，则需要执行提供从现有应用程序客户机项目对侦听器的访问描述的过程。

建立随侦听器一起初始化的应用程序客户机项目

要建立随侦听器一起初始化的应用程序客户机项目，执行下列操作：

1. 单击文件 > 导入。
2. 在“选择”页上，双击应用程序客户机 JAR 文件。
3. 在“应用程序客户机导入”页上，指定几项详细信息：
 - a. 在“应用程序客户机文件”字段中，指定一个 JAR 文件，该文件设置对 TCP/IP 侦听器的访问（但不包含 TCP/IP 侦听器）：

```
installationDir\egl\ eclipse\plugins\  
com.ibm.etools.egl.generators_version\runtime\EGLTcipListener.jar
```

installationDir

产品安装目录，如 C:\Program Files\IBM\RSPD\6.0。如果在安装您要使用的产品之前安装了 Rational Developer 产品并将它保留了下来，则可能需要指定在先前安装中使用的目录。

version

插件的最新版本；例如，6.0.0。

TCP/IP 侦听器本身位于 fdaj6.jar 中，当您第一次将 EGL 代码生成到应用程序客户机项目中时，fdaj.jar 将被放到该项目中。

- b. 单击新建单选按钮，该按钮位于应用程序客户机项目的标签后面。
- c. 在新项目名字段中输入应用程序客户机项目的名称；然后，设置或取消设置使用缺省值复选框。如果设置了此复选框，则项目将存储在用项目名命名的工作空间目录中。如果取消设置了此复选框，则在新项目位置字段中指定项目名。
- d. 指定包含应用程序客户机项目的企业应用程序项目的名称：
 - 如果正在使用现有的 J2EE 1.2 企业应用程序项目，则单击现有单选按钮，该单选按钮位于企业应用程序项目的标签后面。在此例中，在现有项目名字段中指定项目名。
 - 如果正在创建新的企业应用程序项目，则执行下列操作：
 - 1) 单击新建单选按钮，该单选按钮位于企业应用程序项目的标签后面。
 - 2) 在新项目名字段中输入企业应用程序项目的名称。
 - 3) 设置或取消设置使用缺省值复选框。

- 4) 如果设置了此复选框，则项目将存储在用项目名命名的工作空间目录中。
如果取消设置了此复选框，则在**新项目位置**字段中指定项目名。

4. 单击**完成**。
5. 忽略两条警告消息，它们指示当将 EGL 输出生成到项目中时将自动添加 jar 文件（fda6.jar 和 fdaj6.jar）。

在应用程序客户机项目中，部署描述符属性 **tcpiplistener.port** 被设置为侦听器接收数据所用的端口号。缺省情况下，端口号是 9876。要更改端口号，执行下列操作：

1. 在“项目资源管理器”视图中，展开应用程序客户机项目，然后展开 **appClientModule**，再展开 **META-INF**
2. 单击 **application-client.xml > 打开方式 > 部署描述符编辑器**
3. 部署描述符编辑器包括一个“源代码”选项卡；单击该选项卡并更改 9876 值，这是分组中最后一个标记的内容，如下所示：

```
<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. 要保存部署描述符，请按 **Ctrl-S** 键。

提供从现有应用程序客户机项目对侦听器的访问

如果将 EGL 代码生成到不是与侦听器一起初始化的应用程序客户机项目中，则需要更新部署描述符（**application-client.xml**）和清单文件（**MANIFEST.MF**）：

1. 在“项目资源管理器”视图中，展开应用程序客户机项目，然后展开 **appClientModule**，再展开 **META-INF**
2. 单击 **application-client.xml > 打开方式 > 部署描述符编辑器**
3. 部署描述符编辑器包括一个“源代码”选项卡。单击该选项卡。在文本中，紧跟在包含 **<display-name>** 标记的行下面，添加下列条目（但是，如果机器上的端口 9876 已被使用，则用另一个端口号替换 9876）：

```
<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. 要保存部署描述符，请按 **Ctrl-S** 键。
5. 在“项目资源管理器”视图中，单击 **MANIFEST.MF > 打开方式 > JAR 依赖项编辑器**。
6. “JAR 依赖项编辑器”包括一个“依赖项”选项卡。单击该选项卡。
7. 查看“依赖项”部分以确保选择了 **fda6.jar** 和 **fdaj6.jar**。
8. 在“Main 类”部分中，在“Main 类”字段中，输入以下值或使用“浏览”机制来指定以下值：

```
CS0TcpiListenerJ2EE
```

9. 要保存清单文件，请按 **Ctrl-S** 键。

部署应用程序客户机项目

要启动 TCP/IP 侦听器，请执行下列两个过程中的任何一个过程：

- 通过使用 WebSphere 应用程序客户机的启动配置，从工作台中启动侦听器：

1. 切换至 J2EE 透视图

2. 单击 **运行 > 运行**
3. 在“启动配置”页上，单击 **WebSphere V5 应用程序客户机**（如果正在 J2EE 1.3 级别上使用项目，则必须这样做）或 **WebSphere V4 应用程序客户机**
4. 选择现有的配置。另外，也可以单击**新建**并设置配置：
 - a. 在“应用程序”选项卡中，选择企业应用程序项目
 - b. 在“自变量”选项卡中，添加自变量：


```
-CCjar=myJar.jar
```

myJar.jar

应用程序客户机 jar 文件的名称。仅当 ear 文件包含多个客户机 jar 文件时，此自变量才是必需的。在大多数情况下，此值是应用程序客户机项目的名称，后跟扩展名 .jar。

如果您希望确认项目名与 jar 文件名的关系，则执行下列操作：

- 1) 在“项目资源管理器”视图中，展开企业应用程序项目，然后展开 META-INF
 - 2) 单击 **application.xml > 打开方式 > 部署描述符编辑器**。
 - 3) “部署描述符”编辑器包括一个“模块”选项卡。单击该选项卡。
 - 4) 在页的最左边，单击 jar 文件并（在页的最右边）查看与该 jar 文件相关联的项目名。
- 如果已安装 WebSphere Application Server (WAS)，则可以使用 launchClient.bat，此文件位于 WAS 安装目录的子目录 bin 中。

可以按照以下方式从命令提示符调用 launchClient：

```
launchClient myCode.ear -CCjar=myJar.jar
```

myCode.ear

企业归档的名称

myJar.jar

应用程序客户机 jar 文件的名称，如有关工作台过程的内容所述

有关 launchClient.bat 的详细信息，请参阅 WebSphere Application Server 文档。

相关任务

第 330 页的『提供对非 EGL jar 文件的访问』

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

设置 J2EE JDBC 连接

如果要在运行时连接至关系数据库，则需要定义要与程序一起使用的数据库源。WebSphere Sever 管理控制台的帮助系统中提供了指示信息。

定义数据库源时，对下列属性赋值：

JNDI 名称

指定与 JNDI 注册表中与数据库绑定的名称相匹配的值：

- 如果正在定义连接至 J2EE 模块在缺省情况下使用的数据库的数据库源，则确保数据库源定义中指定的 JNDI 名称与在运行时使用的 J2EE 部署描述符中的 **vgj.jdbc.default.database** 属性的值相匹配

- 如果正在定义当系统函数 `VGLib.connectionService` 运行时将访问的数据源，则确保数据源定义中指定的 JNDI 名称与在运行时使用的 J2EE 部署描述符中的适当 `vgj.jdbc.database.SN` 属性值相匹配

数据库名称

指定数据库的名称，数据库管理系统已知此名称

用户标识

指定用于连接至数据库的用户名。

如果数据源定义引用缺省数据库，则在“用户标识”字段中指定的值将在运行时使用的 J2EE 部署描述符的 `vgj.jdbc.default.userid` 属性中设置的任何值覆盖（但仅当为 `vgj.jdbc.default.userid` 和 `vgj.jdbc.default.password` 都指定了值时才会这样）。同样，如果数据源定义引用通过系统函数 `sysLib.connect` 或 `VGLib.connectionService` 访问的数据库，则在“用户标识”字段中指定的值将被调用该系统函数时指定的任何用户标识覆盖（但仅当该调用同时传递用户标识和密码时才会这样）。

可在设置认证别名时指定该名称。要显示定义该别名的屏幕，在“管理控制台”中按如下顺序进行访问：安全性 > 全局安全性 > 认证 > JAAS 配置 > J2C 认证数据。

密码

指定用于连接至数据库的密码。如果数据源定义引用缺省数据库，则在“密码”字段中指定的值将在运行时使用的 J2EE 部署描述符的 `vgj.jdbc.default.password` 属性中设置的值覆盖（但仅当指定了 `vgj.jdbc.default.userid` 和 `vgj.jdbc.default.password` 的值时才会这样）。同样，如果数据源定义引用通过系统函数 `VGLib.connectionService` 访问的数据库，则在“密码”字段中指定的值将被调用该系统函数时指定的任何密码覆盖（但仅当该调用同时传递用户标识和密码时才会这样）。

可在设置认证别名时指定密码。要显示定义该别名的屏幕，在“管理控制台”中按如下顺序进行访问：安全性 > 全局安全性 > 认证 > JAAS 配置 > J2C 认证数据。

可以定义多个数据源，在这种情况下，使用系统函数 `VGLib.connectionService` 来在它们之间切换。

有关部署描述符属性的含义的详细信息（包括有关如何派生生成的值的详细信息），请参阅 *Java 运行时属性*（参考）。

相关任务

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

第 240 页的『了解如何建立标准 JDBC 连接』

相关参考

第 493 页的『Java 运行时属性（详细信息）』

第 510 页的『EGL 中的 JDBC 驱动程序需求』

第 839 页的『`connectionService()`』

部署链接属性文件

链接属性文件必须与使用该文件的 Java 程序位于同一个 J2EE 应用程序中。如果该文件位于该应用程序的顶层目录中，则将 Java 运行时属性 `cso.linkageOptions.LO` 设

置为文件名，并且不指定路径信息。如果该文件位于应用程序顶层目录以下，则使用开始于顶层目录并对每一层包括一个斜线 (/) 的路径，即使应用程序在 Windows 平台上运行也是如此。

在 J2EE 项目中进行开发时，顶层目录与模块所在的项目的 `appClientModule`、`ejbModule` 或 `WebContent` 目录相对应。在开发 Java 项目时，顶层目录在项目目录中。

有关如何定义链接属性文件的格式以及如何标识链接属性文件的其它详细信息，请参阅 [链接属性文件](#)（参考）。

相关概念

第 315 页的『Java 运行时属性』

第 282 页的『链接选项部件』

『链接属性文件』

相关任务

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

第 322 页的『设置部署描述符值』

相关参考

第 370 页的『callLink 元素』

第 86 页的『异常处理』

第 598 页的『链接属性文件（详细信息）』

链接属性文件

链接属性文件是一个文本文件，在 Java 运行时，该文件用来提供有关以下问题的详细信息：生成的 Java 程序或包装器如何调用另一进程中的生成的 Java 程序。

仅当您指定 Java 程序或包装器的链接选项是在运行时而不是在生成时设置的时候，该文件才适用。可以生成该文件，也可以从头开始创建该文件。

有关何时生成该文件以及有关文件格式的详细信息，请参阅 [链接属性文件](#)（详细信息）。有关生成的文件的名称的详细信息，请参阅 [生成的输出](#)（参考）。有关部署的详细信息，请参阅 [部署链接属性文件](#)。

相关概念

第 483 页的『生成的输出』

相关任务

第 329 页的『部署链接属性文件』

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

相关参考

第 484 页的『生成的输出（参考）』

第 358 页的『genProperties』

第 598 页的『链接属性文件（详细信息）』

提供对非 EGL jar 文件的访问

可能需要提供对非 EGL jar 文件的访问，以便调试和运行 EGL 生成的 Java 代码。根据项目类型的不同，提供对那些文件的访问过程也有所不同：

应用程序客户机项目

在使用解释性调试器之前，请在 CLASSPATH 变量中引用非 EGL jar 文件，如设置 *EGL 调试器首选项* 中所述。

在运行代码（使用或不使用 EGL Java 调试器）之前，执行下列操作：

1. 对于每个引用应用程序客户机项目的企业应用程序项目，从文件系统中的目录中导入感兴趣的 jar 文件：
 - a. 在“项目资源管理器”视图中，右键单击企业应用程序项目并单击**导入**
 - b. 在“选择”页中，单击**文件系统**
 - c. 在“文件系统”页中，指定 jar 文件所在的目录
 - d. 在该页右边，选择您感兴趣的 jar 文件
 - e. 单击**完成**
2. 更新应用程序客户机项目中的清单，以使企业应用程序项目中的 jar 文件在运行时可用：
 - a. 在“项目资源管理器”视图中，右键单击应用程序客户机项目并单击**属性**
 - b. 在“属性”页左边，单击 **Java JAR 依赖项**
 - c. 当名为“Java JAR 依赖项”的部分显示在该页右边时，设置每个与您感兴趣的 jar 文件相对应的复选框
 - d. 单击**确定**

EJB 项目

在使用解释性调试器之前，请在 CLASSPATH 变量中引用非 EGL jar 文件，如设置 *EGL 调试器首选项* 中所述。

在运行代码（使用或不使用 EGL Java 调试器）之前，执行下列操作：

1. 对于每个引用 EJB 项目的企业应用程序项目，从文件系统中的目录中导入感兴趣的 jar 文件：
 - a. 在“项目资源管理器”视图中，右键单击企业应用程序项目并单击**导入**
 - b. 在“选择”页中，单击**文件系统**
 - c. 在“文件系统”页中，指定 jar 文件所在的目录
 - d. 在该页右边，选择您感兴趣的 jar 文件
 - e. 单击**完成**
2. 更新 EJB 项目中的清单，以使企业应用程序项目中的 jar 文件在运行时可用：
 - a. 在“项目资源管理器”视图中，右键单击 EJB 项目并单击**属性**
 - b. 在“属性”页左边，单击 **Java JAR 依赖项**
 - c. 当名为“Java JAR 依赖项”的部分显示在该页右边时，设置每个与您感兴趣的 jar 文件相对应的复选框
 - d. 单击**确定**

Java 项目

在通过解释性调试器运行代码之前，请在 CLASSPATH 变量中引用非 EGL jar 文件，如设置 *EGL 调试器首选项* 中所述。

在通过 EGL Java 调试器运行代码之前，对项目的 Java 构建路径添加条目：

1. 在“项目资源管理器”视图中，右键单击 Java 项目并单击**属性**
2. 在“属性”页左边，单击 **Java 构建路径**

3. 当名为“Java 构建路径”的部分显示在该页右边时，单击“库”选项卡
4. 对每个将要添加的 jar 文件，单击**添加外部 Jar** 并使用“浏览”机制来选择文件
5. 要关闭“属性”页，请单击**确定**

J2EE Web 项目

在使用解释性调试器之前，请在 CLASSPATH 变量中引用非 EGL jar 文件，如设置 *EGL 调试器首选项* 中所述。

在运行代码（使用或不使用 EGL Java 调试器）之前，将 jar 文件从文件系统导入到以下 Web 项目文件夹中：

```
WebContent/WEB-INF/lib
```

对于目录中的一组 jar 文件，导入过程如下所示：

1. 在“项目资源管理器”视图中，展开 Web 项目，展开 **WebContent**，展开 **WEB-INF**，右键单击 **lib** 并单击**导入**
2. 在“选择”页中，单击**文件系统**
3. 在“文件系统”页中，指定 jar 文件所在的目录
4. 在该页右边，选择您感兴趣的 jar 文件
5. 单击**完成**

下列 jar 文件需求有效：

- 生成的 Java 程序（无论以任何方式访问 MQSeries）都需要 MQ Series Classes for Java；特别是，Java 程序需要下列 jar 文件（尽管在准备时不需要）：
 - com.ibm.mq.jar
 - com.ibm.mqbind.jar

如果您使用 WebSphere MQ V5.2，则软件位于 IBM WebSphere MQ SupportPac™ MA88 中，后者可以通过访问 IBM Web 站点（www.ibm.com）并搜索 MA88 找到。下载并安装该软件；然后，可以从该软件的安装目录的 JavaLib 子目录中访问 jar 文件。

如果您使用 WebSphere MQ V5.3，则可以通过执行定制安装并选择 Java Messaging 来获取等同的软件。然后，可以从 MQSeries 安装目录的 JavaLib 子目录中访问 jar 文件。

- 生成的 Java 程序或包装器如果使用协议 CICSJ2C 来访问 CICS for z/OS，就需要访问 connector.jar 和 cicsj2ee.jar，但是仅在运行时才如此。那些文件是在您安装 CICS 事务网关时提供给您使用的。

注：当 EGL Java 调试器在 J2EE 中运行时，就可以对 CICS 进行访问。然而，当该调试器在 J2EE 外部运行时，或者当您正在使用 EGL 解释性调试器（它总是在 J2EE 外部运行）时，就会尝试调用 CICS，但会失败。

- 生成的 Java 程序如果访问 SQL 表，就需要随数据库管理系统一起安装的文件：
 - 对于 DB2 UDB，文件是下列其中一项：

```
sql1lib\java\db2java.zip  
sql1lib\java\db2jcc.jar
```

那些文件中的第二个文件对 DB2 UDB 版本 8 或更高版本可用，如 DB2 UDB 文档所述。

- 对于 Informix, 文件如下所示:

```
ifxjdbc.jar  
ifxjdbcx.jar
```

- 对于 Oracle, 请参阅 Oracle 文档。

数据库文件在运行时是必需的, 并且在准备时可以用来验证 SQL 语句。

相关任务

第 106 页的『为 EGL 调试器设置首选项』

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

EGL 引用

EGL 中的赋值兼容性

赋值兼容性规则（如后所述）在下列情况下适用：

- 将一个非引用变量赋给另一个非引用变量时；或者
- EGL 在自变量与函数调用中的相关参数之间传送数据，但仅当接收函数中的参数具有修饰符 IN（在此情况下自变量为源）或 OUT（在此情况下参数为源）时。但是，如果该参数在 PageHandler 的 onPageLoad 函数中，则赋值兼容性规则不适用；有关这一情况的详细信息，请参阅 *EGL 中的引用兼容性*。

赋值兼容性基于以下类型划分：

- 文本类型包括 CHAR、MBCHAR、STRING 和 UNICODE
- 数字类型包括
BIN、INT、BIGINT、SMALLINT、DECIMAL、NUM、NUMBER、FLOAT、SMALLFLOAT 和 MONEY
- 日期时间类型包括 DATE、INTERVAL、TIME 和 TIMESTAMP
- HEX 单独为一个类别
- VisualAge Generator 旧类型包括 DBCHAR、NUMC 和 PACF，每个类型都遵循 VisualAge Generator 规则

赋值兼容性规则如下所示：

- 任何文本类型的字段可以赋值给任何文本类型的字段
- 任何数字类型的字段可以赋值给任何数字类型的字段
- 任何日期时间类型的字段可以赋值给任何文本或数字类型的字段
- 类型为 STRING 或 CHAR 的字段可以赋值给类型为 HEX 的字段，或者类型为 HEX 的字段可以赋值给类型为 STRING 或 CHAR 的字段
- 类型为 CHAR 的字段可以赋值给类型为 NUM 的字段
- 要将数字类型的字段赋值给文本类型的字段，使用系统函数 **StrLib.formatNumber**
- 要将类型为 DATE、TIME 或 TIMESTAMP 的字段赋值给文本类型的格式字段，使用相应的系统函数：
 - **StrLib.formatDate**（用于日期）
 - **StrLib.formatTime**（用于时间）
 - **StrLib.formatTimestamp**（用于时间戳记）
- 要将文本类型的字段赋值给类型为 DATE、TIME 或 TIMESTAMP 文的字段，使用相应的系统函数：
 - **ConverseLib.dateValue**（用于日期）
 - **ConverseLib.timeValue**（用于时间）
 - **ConverseLib.timestampValue**（用于时间戳记）

在数字类型之间赋值

可以将任何数字类型（包括 NUMC 和 PACF）的值赋予具有任何数字类型和大小的字段，并且 EGL 会执行必需的转换来以目标格式保留数据。

根据需要添加或截断无意义的零。值的整数部分中的开头的零是没有意义的，就像值的小数部分中的结尾数字。

对于任何数字类型，可以使用系统变量 `sysVar.overflowIndicator` 来测试赋值或算术计算是否导致了算术溢出，并且可以设置系统变量 `VGVar.handleOverflow` 来指定此类溢出的后果。

如果发生了算术溢出，则目标字段中的值不会更改。如果未发生算术溢出，则根据目标字段的声明来调整赋予目标字段的值。

假定正在将类型为 NUM 的字段复制至另一个字段并且源字段的运行时值为 108.314:

- 如果目标字段允许 7 位数并且有 1 位为小数，则目标字段接收值 000108.3，且不会检测到数字溢出。（不会将小数值中的精度丢失认为是溢出。）
- 如果目标字段允许 4 位数并且有 2 位为小数，则会检测到数字溢出并且目标字段中的值不会更改

对定点类型的字段指定浮点值（类型 FLOAT 或 SMALLFLOAT）时，必要时目标值会被截断。如果源值为 108.357 并且定点目标有一个小数位（如目标接收到 108.3），将出现这种情况。

其它交叉类型赋值

有关其它交叉类型赋值的详细信息如下所示:

- 仅当源声明未包含小数位时，将类型为 NUM 的值赋予类型为 CHAR 的目标才有效。此运算等同于 CHAR 至 CHAR 的赋值。

例如，如果源长度为 4 并且值为 21，则其内容等同于“0021”，并且长度不匹配不会导致错误状态:

- 如果目标的长度为 5，则该值是作为“0021 ”（在右边添加了一个单字节空格）存储的
- 如果目标的长度为 3，则该值是作为“002”（在右边截断了一位）存储的

如果类型为 NUM 的值是负数并且被赋予类型为 CHAR 的值，则复制到字段中的最后一个字节是不可打印字符。

- 仅在下列情况下，将类型为 CHAR 的值赋予类型为 NUM 的目标才有效:
 - 源（字段或文本表达式）包含数字，并且未包含其它字符
 - 目标声明没有小数位

此运算等同于 NUM 至 NUM 的赋值。

例如，如果源长度为 4 并且值为“0021”，则内容等同于数字 21，下列示例显示了长度不匹配的结果:

- 如果目标的长度为 5，则该值是作为 00021（在左边填充了数字零）存储的
- 如果目标的长度为 3，则该值是作为 021（截断了非有效数字）存储的

- 如果目标的长度为 1，则该值将存储为 1
- 可以通过两个步骤来完成从类型为 NUMC 的值到类型为 CHAR 的目标的赋值，如果值为正数，则会除去符号：
 1. 将 NUMC 值赋予类型为 NUM 的目标
 2. 将 NUM 值赋予类型为 CHAR 的目标

如果类型为 NUMC 的目标的值是负数，则复制到类型为 CHAR 的目标中的最后一个字节是不可打印字符。

- 仅当源中的字符位于十六进制数字 (0-9、A-F 和 a-f) 范围内时，将类型为 CHAR 的值赋予类型为 HEX 的目标才有效。
- 将类型为 HEX 的值赋予类型为 CHAR 的目标会将数字和大写字母 (A-F) 存储在目标中。
- 将类型为 MONEY 的值赋予类型为 CHAR 的目标是无效的。将 MONEY 转换为 CHAR 的最好办法是使用系统函数 `strLib.formatNumber`。
- 仅当源值是符合掩码 `yyyyMMdd` 的有效日期时，将类型为 NUM 或 CHAR 的值赋予类型为 DATE 的目标才有效，有关详细信息，请参阅主题 `DATE`。
- 仅当源值是符合掩码 `hhmmss` 的有效时间时，将类型为 NUM 或 CHAR 的值赋予类型为 TIME 的目标才有效，有关详细信息，请参阅主题 `TIME`。
- 仅当源值是符合 `TIMESTAMP` 字段的掩码的有效时间戳记时，将类型为 CHAR 的值赋予类型为 `TIMESTAMP` 的目标才有效。下面是一个示例：

```
// NOT valid because February 30 is not a valid date
myTS timestamp("yyyyMMdd");
myTS = "20050230";
```

如果完整掩码的开头缺少字符（例如，如果掩码为“dd”），则 EGL 假定高级字符（在此情况下为“yyyyMM”）表示当前时刻以符合机器时钟。下列语句导致二月份出现运行时错误：

```
// NOT valid if run in February
myTS timestamp("dd");
myTS = "30";
```

- 将类型为 TIME 或 DATE 的值赋予类型为 NUM 的目标相当于将类型为 NUM 的值赋予类型为 NUM 的目标。
- 将类型为 TIME、DATE 或 `TIMESTAMP` 的值赋予类型为 CHAR 的目标相当于将类型为 CHAR 的值赋予类型为 CHAR 的目标。

字符类型的填充和截断

如果目标具有非 `STRING` 字符类型（包括 `DBCHAR` 和 `HEX`），并具有多于存储源值所需的空空间，则 EGL 会在右边填充数据：

- 使用单字节空格来填充类型为 `CHAR` 或 `MBCHAR` 的目标
- 使用双字节空格来填充类型为 `DBCHAR` 的目标
- 使用 Unicode 双字节空格来填充类型为 `UNICODE` 的目标
- 使用二进制零来填充类型为 `HEX` 的目标，例如，这表示源值“0A”在双字节目标中将存储为“0A00”而不是存储为“000A”

如果字符类型的目标空间不足以存储源值，则 EGL 会截断右边的值。不会发出错误信号。下列情形下可能会发生特殊情况：

- 运行时平台支持 EBCDIC 字符集
- 赋值语句将类型为 MBCHAR 的文字或类型为 MBCHAR 的项复制至类型为 MBCHAR 的较短项
- 逐字节截断将会除去最后的 Shift-in 字符或分割 DBCHAR 字符

在这种情况下，EGL 会按需要截断字符以确保目标项包含类型为 MBCHAR 的有效字符串，然后添加结束单字节空格（如果必要的话）。

时间戳记之间的赋值

如果将类型为 TIMESTAMP 的项赋予另一个类型为 TIMESTAMP 的字段，则下列规则适用：

- 如果源字段的掩码缺少目标字段所需的相对高级的条目，将在赋值时根据机器上的时钟指定这些目标条目，如下列示例所示：

```
- sourceTimeStamp timestamp ("MMdd");
  targetTimeStamp timestamp ("yyyyMMdd");

  sourceTimeStamp = "1201";

  // if this code runs in 2004, the next statement
  // assigns 20041201 to targetTimeStamp
  targetTimeStamp = sourceTimeStamp;
```

```
- sourceTimeStamp02 timestamp ("ssff");
  targetTimeStamp02 timestamp ("mmssff");

  sourceTimeStamp02 = "3201";

  // the next assignment includes the minute
  // that is current when the assignment statement runs
  targetTimeStamp02 = sourceTimeStamp02;
```

- 如果源项的掩码缺少目标字段所需的相对低级的条目，将对这些目标条目指定最低有效值，如下列示例所示：

```
- sourceTimeStamp timestamp ("yyyyMM");
  targetTimeStamp timestamp ("yyyyMMdd");

  sourceTimeStamp = "200412";

  // regardless of the day, the next statement
  // assigns 20041201 to targetTimeStamp
  targetTimeStamp = sourceTimeStamp;
```

```
- sourceTimeStamp02 timestamp ("hh");
  targetTimeStamp02 timestamp ("hhmm");

  sourceTimeStamp02 = "11";

  // regardless of the minute, the next statement
  // assigns 1100 to targetTimeStamp02
  targetTimeStamp02 = sourceTimeStamp02;
```

赋值给具有子结构的字段或从具有子结构的字段进行赋值

可以将具有子结构的字段赋予不具有子结构的字段，反之亦然，并且可以在两个具有子结构的字段之间进行赋值。例如，假定名为 *myNum* 和 *myRecord* 的变量基于下列部件：

```
DataItem myNumPart
  NUM(12)
end
```

```

Record myRecordPart type basicRecord
  10 topMost CHAR(4);
  20 next01 HEX(4);
  20 next02 HEX(4);
end

```

在数学系统变量的外部，将类型为 HEX 的值赋予类型为 NUM 的项无效；但由于 **topMost** 的类型为 CHAR，所以格式为 **myNum = topMost** 的赋值有效。一般来说，赋值语句中的字段的基本类型控制赋值，并且不考虑下级项的基本类型。

缺省情况下，具有子结构的项的基本类型为 CHAR。如果将数据赋予具有子结构的字段或从具有子结构的字段进行数据赋值，并且在声明时未指定另一基本类型，则先前对类型为 CHAR 的字段描述的规则在赋值期间生效。

固定记录的赋值

一条固定记录至另一条固定记录的赋值等同于将一个类型为 CHAR 的具有子结构的项赋予另一个类型为 CHAR 的具有子结构的项。长度不匹配会导致在接收到的值右边添加单字节空格或从接收到的值右边除去单字节字符。赋值不考虑下级结构字段的基本类型。

存在以下例外：

- 可以将记录的内容赋予记录，或赋予类型为 CHAR、HEX 或 MBCHAR 的字段，但是不能赋予任何其它类型的字段
- 记录可以从记录中、从字符串文字中或从 CHAR、HEX 或 MBCHAR 类型的字段中接收数据，但是不能从数字文字中或从除 CHAR、HEX 或 MBCHAR 以外的类型的字段中接收数据

最后，如果将 SQL 记录赋予具有另一类型的记录或者将具有另一类型的记录赋予 SQL 记录，则必须确保非 SQL 记录有空间来存放每个结构字段前面的四字节区域。

相关概念

第 177 页的『PageHandler』

相关参考

第 340 页的『赋值』

第 477 页的『函数参数』

第 481 页的『EGL 源格式的函数部件』

第 619 页的『EGL 源格式的 PageHandler 部件』

第 663 页的『程序参数』

第 664 页的『EGL 源格式的程序部件』

相关参考

第 38 页的『DATE』

第 80 页的『EGL 语句』

第 805 页的『formatNumber()』

第 868 页的『handleOverflow』

第 556 页的『move』

第 854 页的『overflowIndicator』

第 31 页的『基本类型』

第 688 页的『子串』

第 40 页的『TIME』

赋值

EGL 赋值将数据从一个内存区复制至另一个内存区，并且可将数字或文本表达式的结果复制至源字段。

▶▶ `target = source ;` ◀◀

target 字段、记录、固定记录或系统变量。

如果目标字段的类型为 CHAR、DBCHAR 或 UNICODE，可在赋值语句的左边指定子串。将填充子串区域（必要时填充空格），并且被赋值的文本不会超出子串区域而是在必要时截断。有关语法的详细信息，请参阅子串。

source

字段、固定记录或数字或字符表达式

以下是一些赋值示例：

```
z = a + b + c ;
myDate = VGVar.currentShortGregorianDate ;
myUser = sysVar.userID ;
myRecord01 = myRecord02 ;
myRecord02 = "USER" ;
```

EGL 赋值语句的行为与 **move** 语句的行为不同，这在 *move* 中作了描述。

赋值规则在 *EGL* 中的赋值兼容性中作了描述。

相关概念

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 335 页的『EGL 中的赋值兼容性』

第 556 页的『move』

第 688 页的『子串』

关联元素

如资源关联中所述，资源关联部件由关联元素组成。每个元素都特定于一个文件名（属性 第 341 页的『fileName』）并包含一组条目，而每个条目都具有下列属性：

- 第 342 页的『system』
- 第 341 页的『fileType』

system 和 **fileType** 属性的值决定可以使用以下列表中的哪些附加属性：

- 第 341 页的『commit』
- 第 341 页的『conversionTable』
- 第 341 页的『formFeedOnClose』
- 第 342 页的『replace』

- 第 342 页的 『 systemName 』
- 第 342 页的 『 text 』

commit

指定（对于 n iSeries 上 EGL 生成的 Java 程序）是否启用落实控制权。

选择下列其中一个值：

NO（缺省值）

使用 sysLib.commit 或 sysLib.rollback 不起作用。

YES

可以使用 sysLib.commit 和 sysLib.rollback 来定义逻辑工作单元的结束。

conversionTable

指定生成的 Java 程序在访问 MQSeries 消息队列期间使用的转换表的名称。

有关其它信息，请参阅数据转换。

fileType

指定目标系统上的文件组织。可以选择显式类型，如 *seqws*。另外，也可以选择 *default* 值，此值本身就是属性 **fileType** 的缺省值。使用缺省值意味着自动选择文件类型：

- 对于目标系统与 EGL 记录类型的特定组合；或者
- 对于打印输出，当文件名是 *printer* 时。

记录和文件类型交叉引用显示了显式的 **fileType** 值以及当您选择 *default* 时使用的值。

fileName

指一个或多个记录中指定的逻辑文件名。您正在创建一个关联元素，将此名称与一个或多个目标系统上的物理资源关联起来。（对于打印输出，指定 *printer* 值。）

在逻辑文件名中，可以将星号（*）用作全局替换字符；但是，仅当该字符是最后一个字符时才有效。有关详细信息，请参阅资源关联和文件类型。

formFeedOnClose

指示当打印表单的输出结束时是否发出换页指令。（打印表单是在代码发出 **print** 语句时生成的。）

在下列其中一种情况下，仅当 **fileName** 值是 *printer* 时，此属性才可用：

- **system** 值是 *aix*、*iSeriesj* 或 *linux* 并且 **fileType** 值是 *seqws* 或 *spool*；或者
- **system** 值是 *win*，并且 **fileType** 值是 *seqws*。

选择下列其中一个值：

YES

发生换页（缺省值）

NO

不发生换页

replace

指定在将记录添加至文件时是替换文件还是对文件进行追加。仅在下列情况下才使用此条目：

- 正在生成 Java 代码；并且
- 记录具有 **seqws** 文件类型。

选择下列其中一个值：

NO

对文件进行追加（缺省值）

YES

替换文件

system

指定目标平台。选择下列其中一个值：

aix

AIX

iseriesj

iSeries

linux

Linux

win

Windows 2000/NT/XP

任何

任何目标平台；有关详细信息，请参阅[资源关联和文件类型](#)

systemName

指定与文件名相关联的文件或数据集的系统资源名称。如果该值中有空格或下列任何字符，则用单引号或双引号将该值引起来：

% = , () /

text

指定当通过串行记录来访问文件时是否导致生成的 Java 程序执行下列操作：

- 在 **add** 操作期间追加行结束字符。在非 UNIX 平台上，那些字符是回车符和换行符；在 UNIX 平台上，只能是换行符。
- 在 **get** 或 **get next** 操作期间，除去行结束字符。

选择下列其中一个值：

NO

缺省值是不追加或除去行结束字符

YES

进行更改，当生成的程序正在与期望记录以行结束字符结尾的产品交换数据时，此值是有用的。

相关概念

第 277 页的『资源关联和文件类型』

相关任务

第 280 页的『将资源关联部件添加至 EGL 构建文件』

第 281 页的『编辑 EGL 构建文件中的资源关联部件』

第 282 页的『从 EGL 构建文件中除去资源关联部件』

相关参考

第 426 页的『数据转换』

第 490 页的『I/O 错误值』

第 673 页的『记录和文件类型交叉引用』

asynchLink 元素

链接选项部件的 *asynchLink* 元素指定生成的程序如何以异步方式调用另一个程序，这种调用是在起始程序调用系统函数 `sysLib.startTransaction` 时发生的。

如果接受缺省行为，即假定创建的事务将从同一个 Java 包中启动，则无需指定 *asynchLink* 元素。

每个元素都包含 `recordName` 属性，此属性引用一条记录，在正被修改操作的特定 **`sysLib.startTransaction`** 函数中也引用了这个记录。

另一个属性是 **`package`**，仅当被调用程序的源所在的包与调用程序的包不同时，才需要该属性。

相关概念

第 282 页的『链接选项部件』

相关参考

第 344 页的『*asynchLink* 元素中的 `package`』

第 344 页的『*asynchLink* 元素中的 `recordName`』

远程调用的 `csouidpwd.properties` 文件

在下面描述的场景中，必须创建 **`csouidpwd.properties`** 文件并提供对该文件的访问。该文件包括从 Java 程序或包装器进行远程调用所需的认证详细信息。

场景如下所示：

- 链接选项部件的 `callLink` 元素的属性 **`remoteComType`** 设置为 `JAVA400`、`CICSJ2C` 或 `CICSECI`；并且
- 需要用户标识和密码；并且
- 出现下列其中一种情况：
 - 调用是通过 Java 程序进行的，但该代码一开始会用空白值调用系统函数 **`SysLib.setRemoteUser`**；或者

- 调用是通过 Java 包装器进行的，但包括该包装器的 Java 代码使用了空白值调用 CSOCallOptions 方法 **setUserId** 和 **setPassword**。

如果调用 **SysLib.setRemoteUser**（或调用相应的 CSOCallOptions method）时提供的是空白的用户标识或密码，则等效属性的值可在 **csouidpwd.properties** 中找到。

您的任务如下所示：

1. 创建 **csouidpwd.properties** 文件，该文件可以包含格式如下的属性设置，每个设置在单独的一行上：

CSOUID=userid

userid 是远程调用的用户标识

CSOPWD=password

password 是远程调用的密码

2. 确保该文件是由类路径引用的目录。相应目录是项目的 JavaSource 文件夹。

相关概念

第 274 页的『Java 包装器』

相关参考

第 502 页的『Java 包装器类』

第 381 页的『callLink 元素中的 remoteComType』

第 832 页的『setRemoteUser()』

asynchLink 元素中的 package

链接选项部件的 asynchLink 元素的属性 **package** 指定包含正被调用的程序的包的名称。缺省值是调用程序的包。

在生成的 Java 程序中使用的包名是 EGL 程序的包名，但是此包名是小写的；并且，当 EGL 从 asynchLink 元素生成输出时，**package** 的值被更改为（如果有必要的话）小写。

相关概念

第 282 页的『链接选项部件』

相关参考

第 343 页的『asynchLink 元素』

『asynchLink 元素中的 recordName』

asynchLink 元素中的 recordName

链接选项部件的 asynchLink 元素的属性 **recordName** 指定在系统函数 sysLib.startTransaction 中使用的记录的名称。在这种情况下，记录名用来标识与 asynchLink 元素相关联的程序或事务。

在记录名中可以将星号 (*) 用作全局替换字符; 但是, 仅当该字符是最后一个字符时才有效。有关详细信息, 请参阅[链接选项部件](#)。

相关概念

第 282 页的『[链接选项部件](#)』

相关参考

第 343 页的『[asynchLink 元素](#)』

第 344 页的『[asynchLink 元素中的 package](#)』

第 835 页的『[startTransaction\(\)](#)』

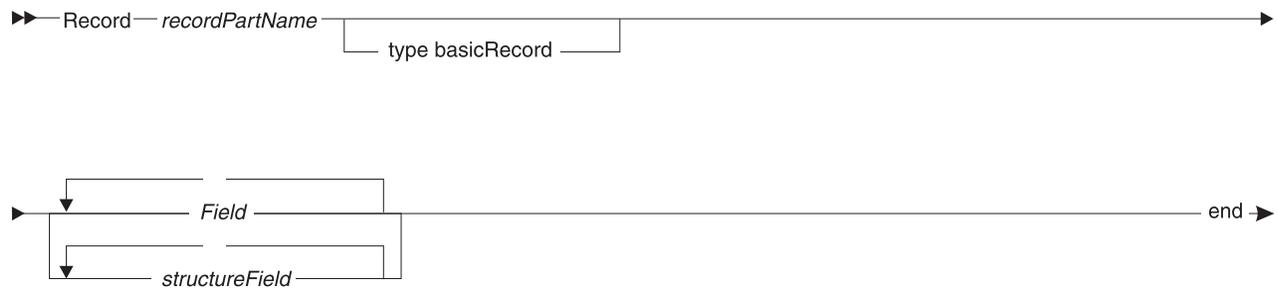
EGL 源格式的基本记录部件

可以在 EGL 文件中声明类型为 `basicRecord` 的记录部件, *EGL* 源格式对该部件作了描述。

基本记录部件的示例如下所示:

```
Record myBasicRecordPart type basicRecord
  10 myField01 CHAR(2);
  10 myField02 CHAR(78);
end
```

基本记录的语法图如下所示:



Record *recordPartName* **basicRecord**

将部件标识为具有 `basicRecord` 类型并指定名称。有关规则, 请参阅[命名约定](#)。

field

记录中的相应变量, 如[记录部件](#)中所述。每个变量声明以分号结束。

structureField

固定结构字段, 如 *EGL* 源格式的结构字段中所述。

相关概念

第 13 页的『[EGL 项目、包和文件](#)』

第 123 页的『[固定记录部件](#)』

第 20 页的『[对部件的引用](#)』

第 16 页的『[部件](#)』

第 122 页的『记录部件』
第 53 页的『引用 EGL 中的变量』
第 25 页的『Typedef』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 431 页的『EGL 源格式的 DataItem 部件』
第 448 页的『EGL 源格式』
第 481 页的『EGL 源格式的函数部件』
第 488 页的『EGL 源格式的带索引记录部件』
第 603 页的『EGL 源格式的 MQ 记录部件』
第 612 页的『命名约定』
第 31 页的『基本类型』
第 664 页的『EGL 源格式的程序部件』
第 673 页的『支持变长记录的属性』
第 676 页的『EGL 源格式的相关记录部件』
第 679 页的『EGL 源格式的串行记录部件』
第 683 页的『EGL 源格式的 SQL 记录部件』
第 686 页的『EGL 源格式中的结构字段』

构建部件

EGL 构建文件格式

.eglbld 文件的结构如下所示:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE EGL PUBLIC "-//IBM//DTD EGL 5.1//EN" "">
<EGL>
  <!-- place your import statements here -->
  <!-- place your parts here -->
</EGL>
```

您的任务是将 import 语句和部件放在 <EGL> 元素内部。

可以指定 <import> 元素来引用包含链中的下一个构建描述符的文件，或者引用构建描述符所引用的任何构建部件。下面是 import 语句的示例:

```
<import file="myBldFile.eglbld"/>
```

可以声明以下列表中的部件:

- <BuildDescriptor>
- <LinkageOptions>
- <ResourceAssociations>

以下是一个简单的示例:

```
<EGL>
  <import file="myBldFile.eglbld"/>
  <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>

```

可以查看构建文件 DTD，它位于以下子目录中：

```

installationDir\egl\eclipse\plugins\
com.ibm.etools.egl_version\dtd

```

installationDir

产品安装目录，如 C:\Program Files\IBM\RSPD\6.0。如果在安装您要使用的产品之前安装了 Rational Developer 产品并将它保留下来，则可能需要指定在之前安装中使用的目录。

version

插件的已安装版本；例如，6.0.0

文件名（如 `egl_wssd_6_0.dtd`）以字母 *egl* 和下划线开头。字符 *wssd* 指的是 Rational Web Developer 和 Rational Application Developer；字符 *wsed* 指的是 Rational Application Developer for z/OS；而字符 *wdsc* 指的是 Rational Application Developer for iSeries。

相关概念

第 30 页的『Import』

第 16 页的『部件』

相关任务

第 118 页的『创建 EGL 源文件』

相关参考

第 441 页的『EGL 编辑器』

构建描述符选项

下表列示了所有构建描述符选项。

构建描述符选项	构建选项过滤器	描述
buildPlan	<ul style="list-style-type: none"> Java 目标 	指定是否创建构建规划
cicsj2cTimeout	<ul style="list-style-type: none"> 调试 Java 目标 Java iSeries 	对 Java 运行时属性 cso.cicsj2c.timeout 赋值，该属性指定在使用 CICSJ2C 协议进行调用期间发生超时之前的毫秒数
commentLevel	<ul style="list-style-type: none"> Java 目标 Java iSeries 	指定将 EGL 系统注释包括在输出源代码中的程度
currencySymbol	<ul style="list-style-type: none"> 调试 Java 目标 	指定由 1 至 3 个字符组成的货币符号

构建描述符选项	构建选项过滤器	描述
dbms	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	指定生成的程序所访问的数据库的类型
decimalSymbol	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	对 Java 运行时属性 vgj.nls.number.decimal 指定一个字符，该运行时属性指示将哪个字符用作十进制符号
destDirectory	<ul style="list-style-type: none"> • Java 目标 	指定用来存储准备输出的目录的名称，但是仅当生成 Java 时才有意义
destHost	<ul style="list-style-type: none"> • Java 目标 	指定构建服务器所在的目标机器的名称或数字 TCP/IP 地址
destPassword	<ul style="list-style-type: none"> • Java 目标 	指定 EGL 用来登录至进行准备的机器的密码
destPort	<ul style="list-style-type: none"> • Java 目标 	指定远程构建服务器侦听构建请求时所用的端口
destUserID	<ul style="list-style-type: none"> • Java 目标 	指定 EGL 用来登录至进行准备的机器的用户标识
eliminateSystemDependentCode	<ul style="list-style-type: none"> • Java 目标 • Java iSeries 	指示 EGL 在验证时是否忽略绝对不会在目标系统中运行的代码。
enableJavaWrapperGen	<ul style="list-style-type: none"> • Java 目标 • Java iSeries 	指定是否允许生成 Java 包装器类
genDataTables	<ul style="list-style-type: none"> • Java 目标 • Java iSeries 	指示是否要生成数据表
genDirectory	<ul style="list-style-type: none"> • Java 目标 	指定一个目录的标准路径，EGL 将把生成的输出和准备状态文件放到该目录中
genFormGroup	<ul style="list-style-type: none"> • Java 目标 	指示是否要生成在正在生成的程序的使用声明中引用的表单组
genHelpFormGroup	<ul style="list-style-type: none"> • Java 目标 	指示是否要生成在正在生成的程序的使用声明中引用的帮助表单组。
genProject	<ul style="list-style-type: none"> • Java 目标 	将 Java 生成输出放到工作台项目中，并自动执行 Java 运行时设置所必需的任务
genProperties	<ul style="list-style-type: none"> • Java 目标 • Java iSeries 	指定要生成的 Java 运行时属性（如果有的话）的类型，并在某些情况下指定是否生成链接属性文件
itemsNullable	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	指定代码可将基本字段设置为 NULL 的情况

构建描述符选项	构建选项过滤器	描述
J2EE	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	指定是否生成要在 J2EE 环境中运行的 Java 程序
linkage	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	包含指导生成的各个方面的链接选项部件的名称
nextBuildDescriptor (请参阅构建描述符部件)	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	标识一个链中的下一个构建描述符
prep	<ul style="list-style-type: none"> • Java 目标 	指定在生成完成且返回码 <= 4 时 EGL 是否开始准备
resourceAssociations	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	包含资源关联部件的名称, 该部件使记录部件与目标平台上的文件和队列相关
sessionBeanID	<ul style="list-style-type: none"> • Java 目标 • Java iSeries 	标识 J2EE 部署描述符中的现有会话元素的名称
sqlCommitControl	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	允许生成一个 Java 运行时属性, 该属性指定每次更改缺省数据库后是否落实更改
sqlDB	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	指定生成的程序所使用的缺省数据库
sqlID	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	指定在 SQL 语句的生成时验证期间或在运行时用来连接至数据库的用户标识
sqlJDBCdriverClass	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	指定在 SQL 语句的生成时验证期间或非 J2EE Java 调试会话期间用来连接至数据库的驱动程序类
sqlJNDIName	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	指定在 J2EE 中运行的生成的 Java 程序所使用的缺省数据库
sqlPassword	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	指定在 SQL 语句的生成时验证期间或运行时用来连接至数据库的密码
sqlValidationConnectionURL	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	指定在 SQL 语句的生成时验证期间用来连接至数据库的 URL

构建描述符选项	构建选项过滤器	描述
system	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	指定生成输出的类别
targetNLS	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	指定用于运行时输出的目标本地语言代码
VAGCompatibility	<ul style="list-style-type: none"> • 调试 • Java 目标 • Java iSeries 	指示生成过程是否允许使用特殊的程序语法
validateSQLStatements	<ul style="list-style-type: none"> • Java 目标 • Java iSeries 	指示是否对数据库验证 SQL 语句

相关概念

第 267 页的『构建描述符部件』

第 315 页的『Java 运行时属性』

相关任务

第 271 页的『将构建描述符部件添加至 EGL 构建文件』

第 272 页的『编辑构建描述符中的一般选项』

相关参考

第 493 页的『Java 运行时属性（详细信息）』

buildPlan

构建描述符选项 **buildPlan** 指定是否创建构建规划。有效值为 YES 和 NO，缺省值为 YES。

构建规划放置在由构建描述符选项 **genDirectory** 标识的目录中。

在将 Java 代码生成到项目中时，会出现特殊情况。所以，无论 **buildPlan** 具有什么设置，都不会创建构建规划，但是出现下列任何一种情况，都会为创建构建规划作准备：

- 每当重建项目时
- 每当生成源文件时；但仅当您选择了工作台首选项在资源修改时自动执行构建时

您可能希望创建构建规划并在稍后的时间调用该规划。有关详细信息，请参阅在生成之后调用构建规划。

相关概念

第 297 页的『构建规划』

相关任务

第 303 页的『在生成之后调用构建规划』

相关参考

第 347 页的『构建描述符选项』

cicsj2cTimeout

当生成 Java 代码时，构建描述符选项 **cicsj2cTimeout** 为 Java 运行时属性 **cso.cicsj2c.timeout** 指定一个值。该属性指定在使用 CICSJ2C 协议进行调用期间发生超时之前的毫秒数。

运行时属性的缺省值为 30000，它表示 30 秒。如果将该值设置为 0，则不会发生超时。该值必须大于等于 0。

当在 WebSphere 390 中运行被调用程序时，属性 **cso.cicsj2c.timeout** 对调用没有任何影响；有关详细信息，请参阅为 *CICSJ2C 调用设置 J2EE 服务器*。

相关概念

第 315 页的『Java 运行时属性』

相关任务

第 325 页的『为 CICSJ2C 调用设置 J2EE 服务器』

相关参考

第 347 页的『构建描述符选项』

第 493 页的『Java 运行时属性（详细信息）』

commentLevel

构建描述符选项 **commentLevel** 指定包含在输出源代码中的 EGL 系统注释的范围。

有效值如下所示：

- 0 输出包含最少的注释，这包括对 EGL 生成的任何名称别名的注释
- 1 除了级别 0 包括的注释外，将脚本编制语句刚好放在生成的用来实现这些语句的代码之前。

缺省值为 1。

提高注释级别对已准备代码的大小或性能没有影响，但会增加输出的大小以及生成、传送和准备输出所需的时间。

相关参考

第 347 页的『构建描述符选项』

currencySymbol

构建描述符选项 **currencySymbol** 仅可用于 Java 输出，并且指定由 1 到 3 个字符组成的货币符号。如果未指定此选项，则缺省值是从生成输出的系统的语言环境派生的。

要指定键盘上所没有的字符，请按住 **Alt** 键并使用数字小键盘来输入字符的十进制代码。例如，在 Windows 2000/NT/XP 上，欧元符号的十进制代码是 0128。

相关概念

第 267 页的『构建描述符部件』

相关参考

第 347 页的『构建描述符选项』

dbms

构建描述符选项 **dbms** 指定生成的程序访问的数据库的类型。选择下列其中一个值:

- DB2 (缺省值)
- INFORMIX
- ORACLE

相关参考

第 347 页的『构建描述符选项』

第 230 页的『Informix 和 EGL』

decimalSymbol

当正在生成 Java 代码时, 构建描述符选项 **decimalSymbol** 将一个字符赋给 Java 运行时属性 **vgj.nls.number.decimal**, 该属性指示用作十进制符号的字符。如果未指定构建描述符选项 **decimalSymbol**, 则该字符由与 Java 运行时属性 **vgj.nls.code** 相关联的语言环境确定。

值只能是 1 个字符。

相关概念

第 315 页的『Java 运行时属性』

相关参考

第 347 页的『构建描述符选项』

第 493 页的『Java 运行时属性 (详细信息)』

destDirectory

构建描述符选项 **destDirectory** 指定用来存储准备输出的目录。仅当生成到目录中而不是生成到项目中时, 此选项才有意义。

当指定标准文件路径时, 除最后一个目录之外的所有目录都必须存在。例如, 如果在 Windows 2000 上指定 `c:\buildout`, 并且如果 `buildout` 目录不存在, 则 EGL 将创建它。如果指定 `c:\interim\buildout` 并且 `interim` 目录不存在, 则准备失败。

如果指定相对目录 (如 USS 上的 `myid/mysource`), 则将把输出置于最底端的目录中, 该目录是相对于缺省目录的, 如后文所述。

destDirectory 的缺省值受构建描述符选项 **destHost** 的状态影响:

- 如果指定了 **destHost**, 则 **destDirectory** 的缺省值是在其中启动构建服务器的目录
- 如果未指定 **destHost**, 则将在发生生成的机器上发生准备, 而 **destDirectory** 的缺省值由构建描述符选项 **genDirectory** 给定

构建描述符选项 **destUserID** 指定的用户必须对接收准备输出的目录具有写权限。

在 USS 上不能使用 UNIX 变量 (例如 `HOME`) 来标识目录结构的部件。

相关参考

第 347 页的『构建描述符选项』

第 353 页的『**destHost**』

第 357 页的『**genProject**』

destHost

构建描述符选项 **destHost** 指定构建服务器所在的目标机器的名称或数字 TCP/IP 地址。没有可用的缺省值。

如果正在准备 Java 输出，则下列描述是适用的：

- **destHost** 是可选的
- 仅当生成到目录中而不是生成到项目中时，**destHost** 才有意义
- 如果指定 **destHost** 而未指定 **destDirectory**，则启动构建服务器时所处的目录就是接收源和准备输出的目录
- 如果未指定 **destHost**，则在发生生成的机器上发生准备；并且，如果未指定 **destDirectory**，则构建描述符选项 **genDirectory** 指定的目录就是接收源和准备输出的目录
- UNIX 环境是区分大小写的

对于名称或 TCP/IP 地址，最多可以输入 64 个字符。如果正在 Windows NT[®] 上进行开发，则必须指定名称而不是 TCP/IP 地址。

destHost 的两个示例值如下所示：

```
abc.def.ghi.com
```

```
9.99.999.99
```

相关参考

第 347 页的『构建描述符选项』

第 352 页的『destDirectory』

『destPassword』

『destPort』

destPassword

构建描述符选项 **destPassword** 指定 EGL 用来登录发生准备的机器的密码。

仅当生成到目录中而不是生成到项目中并且仅当对构建描述符选项 **destHost** 指定了值时，此选项以及本页面上的描述才有意义。

密码为构建描述符选项 **destUserID** 中指定的用户标识提供访问权。对于所有目标系统，密码的值都是区分大小写的。

没有可用的缺省值。

使用 **destPassword** 意味着将密码存储在 EGL 构建文件中。可以通过不设置构建描述符选项来避免安全性风险。启动生成时，可在交互式生成对话框中或在命令行上设置密码。

相关参考

第 347 页的『构建描述符选项』

『destHost』

第 354 页的『destUserID』

destPort

构建描述符选项 **destPort** 指定远程构建服务器侦听构建请求所用的端口。

仅当生成到目录中而不是生成到项目中，并且仅当对构建描述符选项 **destHost** 指定了值时，此选项才有意义。

没有任何缺省值可用。

相关参考

第 347 页的『构建描述符选项』

第 353 页的『destHost』

destUserID

构建描述符选项 **destUserID** 指定 EGL 用来登录发生准备的机器的用户标识。

仅当生成到目录中而不是生成到项目中并且仅当对构建描述符选项 **destHost** 指定了值时，此选项以及本页面上的描述才有意义。

destUserID 指定的用户必须具有写至该目录的权限。对于所有目标系统，选项值都是区分大小写的。

没有可用的缺省值。

相关参考

第 347 页的『构建描述符选项』

第 353 页的『destHost』

第 353 页的『destPassword』

eliminateSystemDependentCode

构建描述符选项 **eliminateSystemDependentCode** 指示 EGL 在验证时是否忽略绝对不会在目标系统中运行的代码。有效值为 *yes*（缺省值）和 *no*。仅当当前生成的输出将在多个系统中运行时，才指定 *no*。

仅当与系统函数 **sysVar.systemType** 相关时，选项 **eliminateSystemDependentCode** 才有意义。该函数本身不影响在生成时验证的代码。例如，即使正在为 Windows 进行生成，也可以验证以下 **add** 语句：

```
if (sysVar.systemType IS AIX)
  add myRecord;
end
```

为了避免验证完全不会在目标系统上运行的代码，请执行下列其中一项操作：

- 将构建描述符选项 **eliminateSystemDependentCode** 设置为 *yes*。在当前示例中，如果将该构建描述符选项设置为 *yes*，则不验证 **add** 语句。但是，您应该了解，仅当逻辑表达式（在此例中，这是 `sysVar.systemType IS AIX`）足够简单从而能够在生成时被求值时，生成器才可以消除对应于系统的代码。
- 另外，将不想验证的语句移至第二个程序；然后，让原始程序有条件地调用新程序：

```
if (sysVar.systemType IS AIX)
  call myAddProgram myRecord;
end
```

相关概念

第 267 页的『构建描述符部件』

相关参考

第 347 页的『构建描述符选项』

enableJavaWrapperGen

当您发出命令以生成程序时，构建描述符选项 **enableJavaWrapperGen** 允许您从下列三个备用项中进行选择：

YES (缺省值)

生成程序并允许生成相关的 Java 包装器类和（如果适当的话）相关的 EJB 会话 bean

ONLY

不生成程序，但允许生成相关的 Java 包装器类和（如果适当的话）相关的 EJB 会话 bean

NO

生成程序，但不生成 Java 包装器类或相关的 EJB 会话 bean（如果有的话）

Java 包装器类和 EJB 会话 bean 的实际生成要求在生成时使用的链接选项部件中进行适当的设置。有关概述，请参阅 *Java 包装器*。

相关概念

第 274 页的『Java 包装器』

相关参考

第 502 页的『Java 包装器类』

genDataTables

构建描述符选项 **genFormGroup** 指示是否要生成由正在生成的程序引用的数据表。这些引用位于程序的使用声明中以及位于程序属性 **msgTablePrefix** 中。

有效值为 *yes*（缺省值）和 *no*。

在下列情况下，将值设置为 *no*：

- 先前已生成了在程序中引用的数据表；并且
- 在上次生成那些表之后未更改它们。

有关其它详细信息，请参阅 *DataTable* 部件。

相关概念

第 267 页的『构建描述符部件』

第 134 页的『DataTable』

相关参考

第 347 页的『构建描述符选项』

第 664 页的『EGL 源格式的程序部件』

第 875 页的『使用声明』

genDirectory

构建描述符选项 **genDirectory** 指定一个目录的标准路径，EGL 将把生成的输出和准备状态文件放到该目录中。

当您正在工作台中生成或者从工作台批处理接口生成时，应遵循下列规则：

对于 Java 生成

必须指定 **genProject** 或 **genDirectory**，但是，如果同时指定这两项，则会产生错误结果。并且，如果为 iSeries 生成代码，则必须指定 **genProject**。

如果正在从 EGL SDK 生成，则应遵循下列规则：

- 必须指定 **genDirectory**
- 如果指定 **genProject**，则会产生错误结果
- 不能为 iSeries 生成 Java 代码

有关部署 Java 代码的详细信息，请参阅处理生成到目录中的 Java 代码。

相关概念

第 302 页的『从 EGL 软件开发包 (SDK) 生成』

第 301 页的『从工作台批处理接口生成』

第 300 页的『工作台中的生成』

相关任务

第 304 页的『处理生成到目录中的 Java 代码』

相关参考

第 347 页的『构建描述符选项』

第 355 页的『genDirectory』

第 357 页的『genProject』

genFormGroup

构建描述符选项 **genFormGroup** 表示是否要生成在正在生成的程序的使用声明中引用的表单组。有效值为 *yes*（缺省值）和 *no*。

帮助表单组（如果有的话）不受此选项影响，但受构建描述符选项 **genHelpFormGroup** 影响。

相关概念

第 267 页的『构建描述符部件』

相关参考

第 347 页的『构建描述符选项』

『genHelpFormGroup』

第 875 页的『使用声明』

genHelpFormGroup

构建描述符选项 **genHelpFormGroup** 表示是否要生成在正在生成的程序的使用声明中引用的帮助表单组。有效值为 *yes*（缺省值）和 *no*。

主表单组不受此选项影响，但受构建描述符选项 **genFormGroup** 影响。

相关概念

第 267 页的『构建描述符部件』

相关参考

第 347 页的『构建描述符选项』

第 356 页的『genFormGroup』

第 875 页的『使用声明』

genProject

构建描述符选项 **genProject** 将 Java 生成输出放入工作台项目，并自动执行 Java 运行时设置所必需的任务。有关该设置以及使用 **genProject** 的好处的详细信息，请参阅在项目中生成 Java 代码。

要使用 **genProject**，请指定项目名。这样，EGL 就忽略构建描述符选项 **buildPlan**、**genDirectory** 和 **prep**，并在下列两种情况的任何一种情况下进行准备：

- 每当重建项目时
- 每当生成源文件时；但仅当您选择了工作台首选项在资源修改时自动执行构建时

如果将选项 **genProject** 设置为工作台中不存在的项目的名称，则 EGL 使用该名称来创建 Java 项目，但是在下列情况下除外：

- 如果要生成 PageHandler 并指定与包含相关 JSP 的项目不同的项目，并且如果那个项目不存在，则 EGL 将创建一个 EGL Web 项目。（但是，建议您将 PageHandler 生成到包含相关 JSP 的项目中。）
- 第二种例外情况涉及 EJB 处理，如果在链接选项部件的 callLink 元素的 **type** 属性为 **ejbCall** 时（对于从包装器到 EGL 生成的程序的调用）生成 Java 包装器，则会发生这种例外情况。在该情况下，EGL 使用 **genProject** 的值来创建 EJB 项目，并使用与 EJB 项目名相同的名称加上字母 **EAR** 来创建新的企业应用程序项目（如果需要的话）。

除了创建项目之外，EGL 还执行下列操作：

- EGL 在项目中创建文件夹。包结构在顶层文件夹 **JavaSource** 下面开始。通过右键单击文件夹名并选择**重构**可以更改名称 **JavaSource**。
- 如果在 Java（已安装的 JRE）的首选项页中指定了 JRE 定义，则 EGL 会添加类路径变量 **JRE_LIB**。该变量包含指向当前使用的 JRE 的运行时 **JAR** 文件的路径。

当您正在工作台中生成或者从工作台批处理接口生成时，应遵循下列规则：

对于 Java 生成

您不需要指定 **genProject** 或 **genDirectory**。如果两个选项都未指定，则将把 Java 输出生成到包含正在生成的 EGL 源文件的项目中。

如果要生成 PageHandler，并且指定的项目存在，则该项目必须是 EGL Web 项目。

如果要生成会话 EJB，并且指定的项目存在，则该项目必须是 EJB 项目。

对于 COBOL 生成

必须指定 **genDirectory**，EGL 将忽略 **genProject** 的任何设置。

如果正在从 EGL SDK 生成，则应遵循下列规则：

- 必须指定 **genDirectory**
- 如果指定 **genProject**，则会产生错误结果
- 不能为 iSeries 生成 Java 代码

相关概念

第 302 页的『从 EGL 软件开发包 (SDK) 生成』

第 301 页的『从工作台批处理接口生成』

第 300 页的『工作台中的生成』

第 293 页的『将 Java 代码生成到项目中』

相关任务

相关参考

第 347 页的『构建描述符选项』

第 350 页的『buildPlan』

第 355 页的『genDirectory』

第 361 页的『prep』

第 385 页的『callLink 元素中的 type』

genProperties

构建描述符选项 **genProperties** 指定要生成的 Java 运行时属性（如果有的话）的类型，并在某些情况下指定是否生成链接属性文件。仅当正在生成 Java 程序（它可以使用任何一种类型的输出）或包装器（它只能使用链接属性文件）时，此构建描述符选项才有意义。

有效值如下所示：

NO（缺省值）

EGL 不生成运行时或链接属性。

PROGRAM

作用如下所示：

- 如果要生成要在 J2EE 外部运行的程序，则 EGL 将生成特定于正在生成的程序的属性文件。该文件的名称如下所示：

```
pgmAlias.properties
```

```
pgmAlias
```

程序在运行时具有的名称。

- 根据您是否指定了 **PROGRAM** 或 **GLOBAL**，还有一些其它作用：
 - 如果要生成在 J2EE 中运行的程序，则 EGL 将生成 J2EE 环境文件或生成到部署描述符中；有关详细信息，请参阅了解设置部署描述符值的备用方法。
 - 如果要生成 Java 包装器或调用程序，则 EGL 可能会生成链接属性文件；有关在什么情况下将会生成此文件的详细信息，请参阅链接属性文件（参考）。

GLOBAL

作用如下所示：

- 如果要生成要在 J2EE 外部运行的程序，则 EGL 将生成一个在整个运行单元中使用的属性文件，但是该属性文件不具有运行单元中的初始程序的名称。属性文件的名称为 **rununit.properties**。

当运行单元的第二个程序不访问文件或数据库，而是调用执行那些操作的程序时，此选项特别有用。

当生成调用程序时，可以生成具有该程序的名称的属性文件，内容可以不包含与数据库相关的属性。当生成被调用程序时，可以生成 **rununit.properties**，内容将可供这两个程序使用。

- 根据您是否指定了 **GLOBAL** 或 **PROGRAM**，还有一些其它作用：
 - 如果要生成在 J2EE 中运行的程序，则 EGL 将生成 J2EE 环境文件或生成到部署描述符中；有关详细信息，请参阅了解设置部署描述符值的备用方法。
 - 如果要生成 Java 包装器或调用程序，则 EGL 可能会生成链接属性文件；有关在什么情况下将会生成此文件的详细信息，请参阅链接属性文件（参考）。

有关更多详细信息，请参阅 *Java 运行时属性和链接属性文件*。

相关概念

- 第 324 页的『J2EE 环境文件』
- 第 315 页的『Java 运行时属性』
- 第 282 页的『链接选项部件』
- 第 330 页的『链接属性文件』

相关任务

- 第 347 页的『构建描述符选项』
- 第 322 页的『设置部署描述符值』

相关参考

- 第 493 页的『Java 运行时属性（详细信息）』

itemsNullable

构建描述符选项 **itemsNullable** 指定代码可将基本字段设置为 NULL 的情况。

有效值如下所示：

NO

不能将基本字段设置为 NULL，但以下情况例外：

- 字段在 SQL 记录中；并且
- SQL 项属性 **isNullable** 设置为 yes。

itemsNullable 的这一设置是缺省值，并且行为与先前版本的 EGL 一致。

YES

可将固定记录以外的任何记录中的任何基本字段设置为 NULL。行为与 Informix 产品 I4GL 一致。

下表显示您所作决定的影响。

表 9. **itemsNullable** 的影响

操作	ItemsNullable = NO	ItemsNullable = YES
对另一字段指定空字段	源的值为 0 或空白，并且赋值会同时复制值和（如果目标是可空的）NULL 状态。	如果目标是可空的，则目标将设置为 NULL。否则，目标将设置为 0 或空白。
在数字表达式中使用空字段	字段被视作包含 0	表达式求值为 NULL
在文本表达式中使用空字段	字段被视作包含空格	字段被视作空字符串

表 9. itemsNullable 的影响 (续)

操作	ItemsNullable = NO	ItemsNullable = YES
在逻辑表达式中使用空字段	表达式被视作字段值为 0 或空白，并且下一示例求值为 TRUE: 0 == null	仅当空值与空值进行比较时，表达式才会求值为 TRUE，下一示例中并非如此，它求值为 FALSE: 0 == null
SET field empty	未设置 Null 状态	已设置 Null 状态
SET record empty	未设置 Null 状态	已设置 Null 状态

相关参考

第 347 页的『构建描述符选项』

J2EE

构建描述符选项 **J2EE** 指定是否生成要在 J2EE 环境中运行的 Java 程序。有效值如下所示:

NO (缺省值)

生成不在 J2EE 环境中运行的程序。该程序直接连接至数据库，并且环境是由属性文件定义的。

YES

生成要在 J2EE 环境中运行的程序。该程序使用数据源连接至数据库，并且环境是由部署描述符定义的。

当生成 PageHandler 时，无论对此选项指定什么内容，J2EE 都始终设置为 YES。

相关概念

第 255 页的『EGL 调试器』

相关参考

第 347 页的『构建描述符选项』

linkage

构建描述符选项 **linkage** 包含链接选项部件的名称，该链接选项部件控制生成的各个方面。此选项并不是生成所必需的，而且没有可用的缺省值。

相关概念

第 282 页的『链接选项部件』

相关参考

第 347 页的『构建描述符选项』

第 370 页的『callLink 元素』

nextBuildDescriptor

构建描述符选项 **nextBuildDescriptor** 标识链中的下一个构建描述符（如果有的话）。有关详细信息，请参阅构建描述符部件。

相关概念

第 267 页的『构建描述符部件』

相关参考

第 347 页的『构建描述符选项』

prep

构建描述符选项 **prep** 指定当生成完成并且返回码 ≤ 4 时 EGL 是否开始准备。有效值为 YES 和 NO，缺省值为 YES。

即使将 **prep** 设置为 NO，以后也可以准备代码。有关详细信息，请参阅在生成之后调用构建规划。

考虑下列情况：

- 将生成到目录中时，EGL 将准备消息写至构建描述符选项 **genDirectory** 所指定的目录并写至结果文件
- 将生成到项目（选项 **genProject**）中时，选项 **prep** 不起任何作用，并在发生下列两种情况中的任何一种时进行准备：
 - 每当重建项目时
 - 每当生成源文件时；但仅当您选择了工作台首选项在资源修改时自动执行构建时

如果想要定制生成的构建规划，执行下列操作：

- 将选项 **prep** 设置为 NO
- 将选项 **buildPlan** 设置为 YES（这是缺省值）
- 生成输出
- 定制构建规划
- 调用构建规划，如 *buildPlan* 所述

相关概念

第 298 页的『结果文件』

相关任务

第 303 页的『在生成之后调用构建规划』

相关参考

第 347 页的『构建描述符选项』

第 350 页的『buildPlan』

第 484 页的『生成的输出（参考）』

第 355 页的『genDirectory』

第 357 页的『genProject』

resourceAssociations

构建描述符选项 **resourceAssociations** 包含资源关联部件的名称，该资源关联部件使记录部件与目标平台上的文件和队列相关联。此选项并不是生成所必需的，而且没有可用的缺省值。

相关概念

第 277 页的『资源关联和文件类型』

相关任务

第 280 页的『将资源关联部件添加至 EGL 构建文件』

相关参考

第 347 页的『构建描述符选项』

第 340 页的『关联元素』

第 673 页的『记录和文件类型交叉引用』

sessionBeanID

构建描述符选项 **sessionBeanID** 标识 J2EE 部署描述符中的现有会话元素的名称。当进行下列操作时，将把环境条目放到会话元素中：

- 为 Java 平台生成程序（通过将 **system** 设置为 AIX、WIN 或 USS）
- 生成到 EJB 项目中（通过将 **genProject** 设置为 EJB 项目）
- 请求生成环境属性（通过将 **genProperties** 设置为 GLOBAL 或 PROGRAM）

选项 **sessionBeanID** 在下列情况中非常有用：

1. 生成 Java 包装器以及 EJB 会话 bean。在 EJB 项目部署描述符（文件 `ejb-jar.xml`）中，EGL 创建会话元素而不带环境条目。

EJB 会话 bean 和会话元素按如下方式命名：

ProgramnameEJBBean

Programname 是通过 EJB 会话 bean 接收数据的运行时程序的名称。名称中的第一个字母是大写的，其它字母是小写的。

在此示例中，程序的名称为 `ProgramA`，而会话元素和 EJB 会话 bean 的名称为 `ProgramaEJBBean`。

2. 生成 EJB 会话 bean 后，将生成 Java 程序本身。因为构建描述符选项 **genProperties** 设置为 YES，所以 EGL 将 J2EE 环境条目生成到部署描述符中，并且是生成到步骤 1 建立的会话元素中。
3. 将生成 `ProgramB`，它是用作 `ProgramA` 的 helper 类的 Java 程序。**system** 和 **genProject** 的值与在步骤 2 中使用的值相同；同样，将生成环境条目并将 **sessionBeanID** 设置为会话元素的名称。

使用 **sessionBeanID** 将导致 EGL 将第二个程序的环境条目放置到步骤 2 中创建的会话元素中，并且是放到会话元素 `ProgramaEJBBean` 中。

在后面的部署描述符部分中，EGL 在生成 `ProgramA` 时在步骤 2 中创建环境条目 **vgj.nls.code** 和 **vgj.nls.number.decimal**；但条目 **vgj.jdbc.default.database** 仅由 `ProgramB` 使用并且是在步骤 3 中创建的：

```
<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>
    </session>
  </enterprise-beans>
</ejb-jar>
```

```

    <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>

```

在可以添加环境条目之前，部署描述符必须包含会话元素。因为会话元素是在 Java 包装器生成期间创建的，所以建议您先生成 Java 包装器，然后再生成相关的程序。

在下列情况下，将程序生成到 EJB 项目中，但将环境条目放到 J2EE 环境文件中而不是放到部署描述符中：

- 已经设置了 **sessionBeanID**，但是在部署描述符中找不到与 **sessionBeanID** 的值相匹配的会话元素；或者
- 未设置 **sessionBeanID**，并且在部署描述符中找不到与该程序同名的会话元素。如果程序是在包装器之前生成的，就会发生这种情况。

对于 EJB 项目，环境条目名（如 **vgj.nls.code**）对每个会话元素只能出现一次。如果环境条目已存在，则 EGL 将更新条目类型和值，而不是创建新条目。

EGL 决不会从部署描述符中删除环境条目。

对于 **sessionBeanID**，没有缺省值可用。

相关参考

第 347 页的『构建描述符选项』

sqlCommitControl

构建描述符选项 **sqlCommitControl** 允许生成 Java 运行时属性，它指定每次更改缺省数据库之后是否落实更改。

仅当构建描述符选项 **genProperties** 也设置为 PROGRAM 或 GLOBAL 时，才生成该属性（**vgj.jdbc.default.database.autoCommit**）。可在部署时设置 Java 运行时属性，不管生成时的决定如何都是如此。

sqlCommitControl 的有效值如下所示：

NOAUTOCOMMIT

落实不是自动进行的；行为与先前版本的 EGL 一致；并且 Java 运行时属性设置为 **false**，这是缺省值。

有关此情况下的落实和回滚的规则的信息，请参阅逻辑工作单元。

AUTOCOMMIT

落实是自动进行的；行为与先前版本的 Informix 产品 I4GL 一致；并且 Java 运行时属性设置为 **true**。

相关概念

第 347 页的『构建描述符选项』

第 315 页的『Java 运行时属性』

相关参考

第 347 页的『构建描述符选项』

第 230 页的『缺省数据库』

第 358 页的『genProperties』

sqlDB

构建描述符选项 **sqlDB** 指定在 J2EE 外部运行的生成的 Java 程序所使用的缺省数据库。值是一个连接 URL；例如，jdbc:db2:MyDB。

选项 **sqlDB** 是区分大小写的，没有缺省值，并且仅当正在生成非 J2EE Java 程序时才使用此选项。此选项对 Java 运行时属性 **vgj.jdbc.default.database** 指定一个值，但仅当将选项 **genProperties** 设置为 GLOBAL 或 PROGRAM 时才如此。

要指定用于验证的数据库，请设置 **sqlValidationConnectionURL**。

相关概念

第 315 页的『Java 运行时属性』

第 209 页的『SQL 支持』

相关参考

第 347 页的『构建描述符选项』

第 358 页的『genProperties』

第 493 页的『Java 运行时属性（详细信息）』

第 366 页的『sqlPassword』

第 366 页的『sqlValidationConnectionURL』

第 365 页的『sqlJDBCdriverClass』

第 369 页的『validateSQLStatements』

sqlID

构建描述符选项 **sqlID** 指定在 SQL 语句的生成时验证期间用来连接至数据库的用户标识。通过设置 **sqlValidationConnectionURL** 来指定数据库。

生成 Java 程序时，EGL 还会将 **sqlID** 的值赋予 Java 运行时属性 **vgj.jdbc.default.userid**。该属性标识在运行时用于连接至缺省数据库的用户标识，并且可以在 **sqlDB** 中指定缺省数据库。

选项 **sqlID** 是区分大小写的，并且没有缺省值。

相关参考

第 347 页的『构建描述符选项』

第 493 页的『Java 运行时属性（详细信息）』

『sqlDB』

第 366 页的『sqlPassword』

第 366 页的『sqlValidationConnectionURL』

第 365 页的『sqlJDBCdriverClass』

第 369 页的『validateSQLStatements』

sqlJDBCDriverClass

构建描述符选项 **sqlJDBCDriverClass** 指定用于连接至数据库的驱动程序类，EGL 在生成时使用它来验证 SQL 语句。通过设置 **sqlValidationConnectionURL** 来指定数据库。数据库访问是通过 JDBC 进行的。

在下列情况下，EGL 还将 **sqlJDBCDriverClass** 的值赋予程序属性文件中的 Java 运行时属性 **vgj.jdbc.drivers**：

- **genProperties** 设置为 GLOBAL 或 PROGRAM
- **J2EE** 设置为 NO

没有任何缺省值可供驱动程序类使用，格式也会根据驱动程序的不同而变化：

- 对于 IBM DB2 APP DRIVER for Windows，驱动程序类如下所示：

```
COM.ibm.db2.jdbc.app.DB2Driver
```

- 对于 IBM DB2 NET DRIVER for Windows，驱动程序类如下所示：

```
COM.ibm.db2.jdbc.net.DB2Driver
```

- 对于 IBM DB2 UNIVERAL DRIVER for Windows，驱动程序类如下所示（com 是小写的）：

```
com.ibm.db2.jcc.DB2Driver
```

- 对于 Oracle JDBC 瘦客户端驱动程序，驱动程序类如下所示：

```
oracle.jdbc.driver.OracleDriver
```

- 对于 IBM Informix JDBC 驱动程序，驱动程序类如下所示：

```
com.informix.jdbc.IfxDriver
```

对于其它驱动程序类，参阅驱动程序的文档。

要指定多个驱动程序类，用冒号（:）将每个类名与下一个类名隔开。如果一个 Java 程序对另一个 Java 程序进行本地调用，但又访问另一数据库管理系统，您可能需要这样做。

相关参考

第 347 页的『构建描述符选项』

第 230 页的『Informix 和 EGL』

第 364 页的『sqlDB』

第 364 页的『sqlID』

第 366 页的『sqlPassword』

第 366 页的『sqlValidationConnectionURL』

第 369 页的『validateSQLStatements』

sqlJNDIName

构建描述符选项 **sqlJNDIName** 指定在 J2EE 中运行的生成的 Java 程序所使用的缺省数据库。该值是在 JNDI 注册表中与缺省数据源绑定的名称；例如，jdbc/MyDB。

选项 **sqlJNDIName** 是区分大小写的，没有缺省值，并且仅当正在为 J2EE 生成 Java 程序时才使用此选项。此选项对 Java 运行时属性 **vgj.jdbc.default.database** 指定一个值，但仅当将选项 **genProperties** 设置为 GLOBAL 或 PROGRAM 时才如此。

要指定用于验证的数据库，请设置 **sqlValidationConnectionURL**。

相关概念

第 315 页的『Java 运行时属性』

第 209 页的『SQL 支持』

相关参考

第 347 页的『构建描述符选项』

第 358 页的『genProperties』

第 493 页的『Java 运行时属性（详细信息）』

『sqlPassword』

『sqlValidationConnectionURL』

第 365 页的『sqlJDBCdriverClass』

第 369 页的『validateSQLStatements』

sqlPassword

构建描述符选项 **sqlPassword** 指定在 SQL 语句的生成时验证期间用来连接至数据库的密码。通过设置 **sqlValidationConnectionURL** 来指定数据库。

生成 Java 程序时，EGL 还会将 **sqlPassword** 的值赋予 Java 运行时属性 **vgj.jdbc.default.password**。该属性标识在运行时用于连接至缺省数据库的密码，并且可以在 **sqlDB** 中指定缺省数据库。

选项 **sqlPassword** 是区分大小写的，并且没有缺省值。

相关概念

第 315 页的『Java 运行时属性』

相关参考

第 347 页的『构建描述符选项』

第 493 页的『Java 运行时属性（详细信息）』

第 364 页的『sqlDB』

第 364 页的『sqlID』

『sqlValidationConnectionURL』

第 365 页的『sqlJDBCdriverClass』

第 369 页的『validateSQLStatements』

sqlValidationConnectionURL

构建描述符选项 **sqlValidationConnectionURL** 指定用来连接至数据库的 URL，EGL 在生成时使用它来验证 SQL 语句。数据库访问是通过 JDBC 进行的。

没有任何缺省值可供 URL 使用，格式也会根据驱动程序的不同而变化：

- 对于 IBM DB2 APP DRIVER for Windows，URL 如下所示：

```
jdbc:db2:dbName
```

dbName

数据库名称

- 对于 Oracle JDBC 瘦客户端驱动程序，URL 根据数据库位置而变化。如果数据库相对于机器来说是本地的，则 URL 如下所示：

```
jdbc:oracle:thin:dbName
```

如果数据库在远程服务器上，则 URL 如下所示：

`jdbc:oracle:thin:@host:port:dbName`

host

数据库服务器的主机名

port

端口号

dbName

数据库名称

- 对于 IBM Informix JDBC 驱动程序，URL 如下所示（应将下列各行组成一行）：

```
jdbc:informix-sqli://host:port  
/dbName:informixserver=servername;  
user=userName;password=password
```

host

数据库服务器所在的机器的名称

port

端口号

dbName

数据库名称

serverName

数据库服务器的名称

userName

Informix 用户标识

password

与用户标识相关联的密码

- 对于其它驱动程序，参阅驱动程序的文档。

相关参考

第 347 页的『构建描述符选项』

第 230 页的『Informix 和 EGL』

第 364 页的『sqlDB』

第 364 页的『sqlID』

第 366 页的『sqlPassword』

第 365 页的『sqlJDBCdriverClass』

第 369 页的『validateSQLstatements』

system

构建描述符选项 **system** 指定目标生成平台。此选项是必需的；没有任何缺省值可用。有效值如下所示：

AIX

指示生成操作将生成可以在 AIX 上运行的 Java 程序

ISERIESJ

指示生成操作将生成可以在 iSeries 上运行的 Java 程序

LINUX

指示生成操作将生成可以在 Linux（带有 Intel 处理器）上运行的 Java 程序

USS

指示生成操作将生成可以在 z/OS UNIX System Services 上运行的 Java 程序

WIN

指示生成操作将生成可以在 Windows 2000/NT/XP 上运行的 Java 程序

相关概念

第 483 页的『生成的输出』

第 282 页的『链接选项部件』

第 8 页的『运行时配置』

相关参考

第 347 页的『构建描述符选项』

第 370 页的『callLink 元素』

第 484 页的『生成的输出（参考）』

第 230 页的『Informix 和 EGL』

targetNLS

构建描述符选项 **targetNLS** 指定用来标识运行时消息的本地语言代码。

下表列示了受支持的语言。在目标平台上，必须装入指定的语言的代码页。

代码	语言
CHS	简体中文
CHT	繁体中文
DES	瑞士德语
DEU	德语
ENP	大写英语（在 Windows 2000、Windows NT 和 z/OS UNIX System Services 上不受支持）
ENU	美国英语
ESP	西班牙语
FRA	法语
ITA	意大利语
JPN	日语
KOR	韩国语
PTB	巴西葡萄牙语

EGL 确定开发机器上的 Java 语言环境是否与其中一种受支持语言相关联。如果回答为“是”，则 **targetNLS** 的缺省值是受支持的语言。否则，**targetNLS** 没有任何缺省值。

相关参考

第 347 页的『构建描述符选项』

VAGCompatibility

构建描述符选项 **VAGCompatibility** 指示生成过程是否允许使用特殊的程序语法，如与 *VisualAge Generator* 的兼容性中所述。有效值是 *no* 和 *yes*。

EGL 首选项 **VAGCompatibility** 的设置确定了此构建描述符的缺省值。如果正在 EGL SDK 中生成，则没有首选项可用，并且 **VAGCompatibility** 的缺省值是 *no*。

仅当程序或 PageHandler 使用特殊的语法时，才指定 *yes*。

相关概念

第 267 页的『构建描述符部件』

第 400 页的『与 VisualAge Generator 的兼容性』

相关参考

第 347 页的『构建描述符选项』

validateSQLStatements

构建描述符选项 **validateSQLStatements** 指示是否对数据库验证 SQL 语句。要成功地使用 **validateSQLStatements**，您需要指定选项 **sqlValidationConnectionURL** 以及（在大多数情况下）以字母 **sql** 开头的其它选项，如稍后所列示的那样。

有效值为 YES 和 NO，缺省值为 NO。SQL 语句的验证增加了生成代码所需的时间。

请求 SQL 验证时，从生成平台访问的数据库管理器将动态准备 SQL 语句。

SQL 语句验证有以下限制：

- 对于使用动态 SQL 且基于 SQL 记录的 SQL 语句，不能进行验证
- 验证过程可能指示数据库管理器在生成环境中发现的错误，而数据库管理器将不会在目标平台上发现这些错误
- 仅当 JDBC 驱动程序支持 SQL prepare 语句的验证并且（在某些情况下）仅当已将驱动程序配置为执行此类验证时，才会进行验证；有关详细信息，请参阅 JDBC 驱动程序的文档

相关参考

第 347 页的『构建描述符选项』

第 364 页的『sqlID』

第 366 页的『sqlPassword』

第 366 页的『sqlValidationConnectionURL』

第 365 页的『sqlJDBCdriverClass』

构建脚本

EGL 构建脚本中的必需选项

如果正在使用 DB2 UDB，则在 EGL 构建脚本中，某些准备选项是必需的。

DB2 预编译器的必需选项

下列选项是使用 DB2 所必需的，并且包括在 fdaptcl 构建脚本中：

- HOST(COB2)
- APOSTSQL
- QUOTE

callLink 元素

链接选项部件的 `callLink` 元素指定在进行调用时使用的链接类型。每个元素都包括下列属性:

- `pgmName`
- `type`

`type` 属性的值确定哪些附加属性可用, 如下列各节所示:

- 『如果 `callLink` 类型是 `localCall` (缺省值)』
- 『如果 `callLink` 类型是 `remoteCall`』
- 第 371 页的 『如果 `callLink` 类型是 `ejbCall`』

如果 `callLink` 类型是 `localCall` (缺省值)

当正在生成 Java 程序, 并且该程序调用位于同一线程中的生成的 Java 程序时, 将属性 `type` 设置为 `localCall`。在这种情况下, 不使用 EGL 中间件, 并且对于其中的 `pgmName` 标识了被调用程序的 `callLink` 元素, 下列属性有意义:

- 第 372 页的 『`callLink` 元素中的 `alias`』
- 第 378 页的 『`callLink` 元素中的 `package`』
- 第 379 页的 『`callLink` 元素中的 `pgmName`』
- 第 385 页的 『`callLink` 元素中的 `type`』

如果被调用程序与调用程序位于同一个包中, 并且如果下列任何一个条件生效, 则不需要为调用指定 `callLink` 元素:

- 未指定被调用程序的外部名; 或者
- 被调用程序的外部名与该程序的部件名完全相同。

当正在生成 Java 包装器时, `type` 的值不能是 `localCall`。

如果 `callLink` 类型是 `remoteCall`

当正在生成 Java 程序或包装器, 并且 Java 代码调用在另一线程中运行的程序时, 将属性 `type` 设置为 `remoteCall`。调用不是通过生成的 EJB 会话 bean 进行的。在这种情况下, 使用 EGL 中间件, 并且对于其中的 `pgmName` 标识了被调用程序的 `callLink` 元素, 下列属性有意义:

- 第 372 页的 『`callLink` 元素中的 `alias`』
- 第 372 页的 『`callLink` 元素中的 `conversionTable`』
- 第 376 页的 『`callLink` 元素中的 `location`』
- 第 378 页的 『`callLink` 元素中的 `package`』 (仅当生成的代码正在调用存储在另一个包中的 Java 程序时才使用)
- 第 379 页的 『`callLink` 元素中的 `pgmName`』
- 第 381 页的 『`callLink` 元素中的 `remoteBind`』
- 第 381 页的 『`callLink` 元素中的 `remoteComType`』
- 第 383 页的 『`callLink` 元素中的 `remotePgmType`』
- 第 384 页的 『`callLink` 元素中的 `serverID`』
- 第 385 页的 『`callLink` 元素中的 `type`』

如果 callLink 类型是.ejbCall

当需要 callLink 元素来处理下列任何一种情况时，将属性 **type** 设置为 `ejbCall`：

- 正在生成 Java 包装器并且要通过生成的 EJB 会话 bean 调用相关的生成的程序
- 正在生成 Java 程序并且要通过生成的 EJB 会话 bean 调用另一个生成的程序

在这种情况下，使用 EGL 中间件，并且对于其中的 **pgmName** 标识了被调用程序的 callLink 元素，下列属性有意义：

- 第 372 页的『callLink 元素中的 alias』
- 第 372 页的『callLink 元素中的 conversionTable』
- 第 376 页的『callLink 元素中的 location』
- 第 378 页的『callLink 元素中的 package』（仅当生成的 Java 代码正在调用一个 Java 程序，而存储该程序的包与 EJB 会话 bean 所在的包不同时才使用）
- 第 378 页的『callLink 元素中的 parmForm』（仅当生成的 Java 代码正在调用在 CICS 上运行的程序时才使用）
- 第 379 页的『callLink 元素中的 pgmName』
- 第 379 页的『callLink 元素中的 providerURL』
- 第 381 页的『callLink 元素中的 remoteBind』
- 第 381 页的『callLink 元素中的 remoteComType』
- 第 383 页的『callLink 元素中的 remotePgmType』
- 第 384 页的『callLink 元素中的 serverID』
- 第 385 页的『callLink 元素中的 type』

相关概念

第 282 页的『链接选项部件』

第 8 页的『运行时配置』

相关任务

第 285 页的『编辑链接选项部件的 callLink 元素』

相关参考

第 372 页的『callLink 元素中的 alias』

第 372 页的『callLink 元素中的 conversionTable』

第 375 页的『callLink 元素中的 linkType』

第 376 页的『callLink 元素中的 location』

第 378 页的『callLink 元素中的 package』

第 379 页的『callLink 元素中的 pgmName』

第 379 页的『callLink 元素中的 providerURL』

第 381 页的『callLink 元素中的 remoteBind』

第 381 页的『callLink 元素中的 remoteComType』

第 383 页的『callLink 元素中的 remotePgmType』

第 384 页的『callLink 元素中的 serverID』

第 385 页的『callLink 元素中的 type』

callLink 元素中的 alias

链接选项部件的 callLink 元素的属性 **alias** 指定在属性 **pgmName** 中标识的程序的运行时名称。仅当 **pgmName** 是指由正在生成程序调用的程序时，该属性才有意义。

此属性的值必须与声明程序时指定的别名（如果有的话）相匹配。如果在声明程序时未指定别名，则将 callLink 元素属性 **alias** 设置为程序部件的名称或者根本不设置该属性。

相关概念

第 282 页的『链接选项部件』

相关任务

第 285 页的『编辑链接选项部件的 callLink 元素』

相关参考

第 370 页的『callLink 元素』

第 379 页的『callLink 元素中的 pgmName』

callLink 元素中的 conversionTable

链接选项部件的 callLink 元素的属性 **conversionTable** 指定用来在调用时转换数据的转换表的名称。仅当 **pgmName** 标识了由生成的程序或包装器调用的程序时，此属性才有意义。

下列详细信息有效:

- 当对非 Java 程序进行调用时，将根据在调用平台上使用的字符集（ASCII 或 EBCDIC）进行缺省转换。在以下情况下，必须指定 **conversionTable** 的值：
 - 调用程序是 Java 代码并且位于支持一种字符集（EBCDIC 或 ASCII）的机器上；并且
 - 被调用程序是非 Java 程序，并且位于支持其它字符集的机器上。
- 当 EGL 生成的 Java 代码调用 Java 程序时，尝试指定转换表将不起作用，但是双向文本的情况除外。
- 仅当属性 **type** 的值是 `ejbCall` 或 `remoteCall` 时，属性 **conversionTable** 才可用。

选择下列其中一个值:

转换表名

调用程序使用指定的转换表。有关表的列表，请参阅数据转换。

- * 使用缺省转换表。选择的表基于客户机的语言环境或者（如果客户机正在 Web 应用程序服务器上运行的话）基于该服务器的语言环境。如果找到不识别的语言环境，则采用英语。

有关表的列表，请参阅数据转换。

programControlled

在运行时，调用程序使用系统项 `sysVar.callConversionTable` 中的转换表名。如果 `sysVar.callConversionTable` 包含空白，则不进行任何转换。

相关概念

第 282 页的『链接选项部件』

相关任务

第 285 页的『编辑链接选项部件的 callLink 元素』

相关参考

第 429 页的『双向语言文本』

第 370 页的『callLink 元素』

第 426 页的『数据转换』

第 379 页的『callLink 元素中的 pgmName』

第 822 页的『convert()』

第 368 页的『targetNLS』

第 385 页的『callLink 元素中的 type』

callLink 元素中的 ctgKeyStore

链接选项部件的 callLink 元素的 **ctgKeyStore** 属性是通过 Java 工具 keytool.exe 或通过 CICS 事务网关工具 IKEYMAN 生成的密钥库的名称。当 **remoteComType** 属性值设置为 CICSSSL 时，此属性是必需的。

相关概念

第 282 页的『链接选项部件』

相关参考

第 370 页的『callLink 元素』

『callLink 元素中的 ctgKeyStorePassword』

第 381 页的『callLink 元素中的 remoteComType』

callLink 元素中的 ctgKeyStorePassword

链接选项部件的 callLink 元素的属性 **ctgKeyStorePassword** 是生成密钥库时使用的密码。

相关概念

第 282 页的『链接选项部件』

相关参考

第 370 页的『callLink 元素』

『callLink 元素中的 ctgKeyStore』

第 381 页的『callLink 元素中的 remoteComType』

callLink 元素中的 ctgLocation

链接选项部件的 callLink 元素的属性 **ctgLocation** 是用于访问 CICS 事务网关 (CTG) 服务器的 URL，此属性是当属性 **remoteComType** 的值是 CICSECI 或 CICSSSL 时使用的。通过设置属性 **ctgPort** 来指定相关端口。

相关概念

第 282 页的『链接选项部件』

相关参考

第 370 页的『callLink 元素』

第 381 页的『callLink 元素中的 remoteComType』

callLink 元素中的 ctgPort

链接选项部件的 callLink 元素的属性 **ctgPort** 是用于访问 CICS 事务网关 (CTG) 服务器的端口, 此属性是当属性 **remoteComType** 的值是 **CICSECI** 或 **CICSSSL** 时使用的。通过设置属性 **ctgLocation** 来指定相关 URL。

对于 **CICSSSL**, **ctgPort** 的值是 CTG JSSE 侦听器侦听请求时所用的 TCP/IP 端口; 如果未指定 **ctgPort**, 则使用 CTG 缺省端口 8050。

相关概念

第 282 页的『链接选项部件』

相关参考

第 370 页的『callLink 元素』

第 373 页的『callLink 元素中的 ctgLocation』

第 381 页的『callLink 元素中的 remoteComType』

callLink 元素中的 JavaWrapper

链接选项部件的 **callLink** 元素的属性 **javaWrapper** 表示是否允许生成 Java 包装器类, 该包装器类可以调用正在生成的程序。

有效值如下所示:

No (缺省值)

不允许生成 Java 包装器类。

Yes

允许生成发生。仅当构建描述符选项 **enableJavaWrapperGen** 设置为 **yes** 或 **only** 时, 才发生生成。

仅当您正在设置远程调用时 (当 **callLink** 的 **type** 属性值是 **remoteCall** 时, 便是这种情况), 您对 **javaWrapper** 属性所作的选择才起作用。相反, 如果正在通过 EJB 设置对程序的调用, 则 **javaWrapper** 的值始终是 **yes**; 如果正在设置本地调用, 则 **javaWrapper** 的值始终是 **no**。

如果正在工作台中生成或从工作台批处理接口生成, 则构建描述符选项 **genProject** 标识接收类的项目。如果未指定 **genProject** (或者, 如果正在 EGL SDK 中生成), 则包装器类被放在由构建描述符选项 **genDirectory** 指定的目录中。

相关概念

第 282 页的『链接选项部件』

相关参考

第 370 页的『callLink 元素』

第 355 页的『genDirectory』

第 357 页的『genProject』

callLink 元素中的 linkType

当属性 **type** 的值是 `localCall` 时，链接选项部件 `callLink` 元素的属性 **linkType** 指定链接的类型。

如果要生成 Java 程序，则当属性 **pgmName** 是指由正在生成的程序调用的程序时，**linkType** 有意义。如果要生成 Java 包装器，则属性 `type` 必须是 `remoteCall` 或 `ejbCall`，且 **linkType** 不可用。

从以下列表中选择一个值：

DYNAMIC

指定调用的是同一个线程中的 Java 程序。DYNAMIC 是缺省值。

STATIC

STATIC 相当于 DYNAMIC，除非正在使用 Rational Application Developer for iSeries 或 Rational Application Developer for z/OS。

相关概念

第 282 页的『链接选项部件』

相关任务

第 285 页的『编辑链接选项部件的 callLink 元素』

相关参考

第 370 页的『callLink 元素』

第 379 页的『callLink 元素中的 pgmName』

第 385 页的『callLink 元素中的 type』

callLink 元素中的 library

当 **type** 属性的值是 `ejbCall` 或 `remoteCall` 时，链接选项部件的 `callLink` 元素的属性 **library** 指定一个 DLL 或库，该 DLL 或库包含被调用程序：

- 如果 EGL 生成的 Java 程序正在调用 iSeries 上非 EGL 生成的远程程序（例如，C 或 C++ 服务程序），则被调用程序属于 iSeries 库，并且 **library** 属性指的是包含将要调用的入口点的程序的名称。设置其它 `callLink` 属性，如下所示：
 - 将 **pgmName** 属性设置为入口点的名称
 - 将 **remoteComType** 属性设置为 `direct` 或 `distinct`
 - 将 **remotePgmType** 属性设置为 `externallyDefined`
 - 将 **location** 属性设置为 iSeries 库的名称
- 否则，如果调用程序是不在 iSeries 上的 EGL 生成的 Java 程序，则 **library** 属性指的是一个 DLL 的名称，该 DLL 包含将要在本地作为本机程序被调用的入口点。该入口点由 **pgmName** 属性标识；但是，仅当入口点与 DLL 具有不同的名称时才需要指定 **library** 属性。

要调用本机 DLL，请按如下方式设置其它 `callLink` 属性：

- 将 **remoteComType** 属性设置为 `direct`
- 将 **remotePgmType** 属性设置为 `externallyDefined`

- 将 **type** 属性设置为 `remoteCall`，这是因为即使在运行 Java 程序的机器上调用 DLL 也将使用 EGL 中间件。

相关概念

第 282 页的『链接选项部件』

相关参考

第 370 页的『callLink 元素』

callLink 元素中的 location

链接选项部件的 `callLink` 元素的属性 **location** 指定在运行时如何确定被调用程序的位置。属性 **location** 适用于以下情况:

- 属性 **type** 的值为 `ejbCall` 或 `remoteCall`;
- 属性 **remoteComType** 的值是 `JAVA400`、`CICSECI`、`CICSSSL`、`CICSJ2C` 或 `TCPIP`;
并且
- 出现下列其中一种情况:
 - 如果要生成 Java 程序，则属性 **pgmName** 是指由正在生成的程序调用的程序
 - 如果要生成 Java 包装器，则 **pgmName** 是指通过 Java 包装器调用的程序

从以下列表中选择一个值:

programControlled

指定当进行调用时，被调用程序的位置是从系统函数 `sysVar.remoteSystemID` 获取的。

system name

指定被调用程序所在的位置。

如果要生成 Java 程序或包装器，则此属性的含义取决于属性 **remoteComType**:

- 如果 **remoteComType** 的值是 `JAVA400`，则 **location** 指的是 `iSeries` 系统标识
- 如果 **remoteComType** 的值是 `CICSECI` 或 `CICSSSL`，则 **location** 指的是 `CICS` 系统标识
- 如果 **remoteComType** 的值为 `CICSJ2C`，则 **location** 是指为 `call` 语句调用的 `CICS` 事务建立的 `ConnectionFactory` 对象的 `JNDI` 名称。当设置 `J2EE` 服务器时，建立该 `ConnectionFactory` 对象，如为 `CICSJ2C` 调用设置 `J2EE` 服务器中所述。按照惯例，`ConnectionFactory` 对象的名称以 `eis/` 开头，如以下示例所示：
`eis/CICS1`
- 如果 **remoteComType** 的值为 `TCPIP`，则 **location** 是指 `TCP/IP` 主机名，并且不存在缺省值
- 如果以下所有条件都成立，则 **location** 是指被调用程序的库：
 - 被调用程序是 `EGL` 生成的 Java 程序，它在 `iSeries` 本地运行
 - **remoteComType** 值是 `DIRECT` 或 `DISTINCT`
 - **remotePgmType** 值是 `EXTERNALLYDEFINED`

相关概念

第 282 页的『链接选项部件』

相关任务

第 285 页的『编辑链接选项部件的 callLink 元素』
第 325 页的『为 CICSJ2C 调用设置 J2EE 服务器』

相关参考

第 370 页的『callLink 元素』

第 379 页的『callLink 元素中的 pgmName』
第 381 页的『callLink 元素中的 remoteComType』
第 385 页的『callLink 元素中的 type』

callLink 元素中的 luwControl

链接选项部件的 callLink 元素的属性 **luwControl** 指定是调用程序还是被调用程序控制工作单元。仅在以下情况下，此属性才适用：

- 属性 **type** 的值为 remoteCall；并且
- 要生成 Java 程序或包装器：
 - 如果正在生成 Java 程序，则属性 **pgmName** 是指由正在生成的程序调用的基于 CICS 的程序
 - 如果正在生成 Java 包装器，则 **pgmName** 是指通过 Java 包装器调用的基于 CICS 的程序

选择下列其中一个值：

CLIENT

指定工作单元受调用程序控制。在调用程序请求落实或回滚之前，不会落实或回滚由被调用程序所作的更新。如果被调用程序发出落实或回滚，则会发生运行时错误。

CLIENT 是缺省值，除非调用程序控制的工作单元在被调用程序所在的平台上不受支持。

如果调用程序是通过 IBM Toolbox for Java 与基于 iSeries 的 COBOL 程序通信的 Java 包装器或程序，则 CLIENT 可用。在这种情况下，该调用的 **remoteComType** 值是 JAVA400。

SERVER

指定被调用程序所启动的工作单元与调用程序所控制的任何工作单元无关。在被调用程序中，下列规则适用：

- 对可恢复的资源所作的第一次更改将开始一个工作单元
- 系统函数 sysLib.commit 和 sysLib.rollback 的使用是有效的

当从 EGL 生成的 Java 代码调用 VisualAge Generator COBOL 程序时，被调用程序返回时将自动发出落实（如果异常终止，就会自动发出回滚）。该命令只会影响由被调用程序所作的更改。

当属性 **type** 是.ejbCall 时，运行时行为与对 SERVER 描述的运行时行为相同。

相关概念

第 282 页的『链接选项部件』
第 279 页的『逻辑工作单元』

相关任务

第 285 页的『编辑链接选项部件的 callLink 元素』

相关参考

第 370 页的『callLink 元素』

第 818 页的『commit()』

第 830 页的『rollback()』

第 379 页的『callLink 元素中的 pgmName』

第 385 页的『callLink 元素中的 type』

callLink 元素中的 package

链接选项部件的 callLink 元素的属性 **package** 标识被调用 Java 程序所在的 Java 包。无论 **type** 属性是 `ejbcall`、`localCall` 还是 `remoteCall`，此属性都有用。

如果要生成 Java 程序，则当属性 **pgmName** 是指由正在生成的程序调用的程序时，**package** 有意义。如果要生成 Java 包装器，则当属性 **pgmName** 是指通过 Java 包装器调用的程序时，**package** 有意义。

如果未指定 **package** 属性，则假定被调用程序与调用程序位于同一个包中。

在生成的 Java 程序中使用的包名是 EGL 程序的包名，但是此包名是小写的；并且，当 EGL 从 callLink 元素生成输出时，**package** 的值被更改为（如果有必要的话）小写。

相关概念

第 282 页的『链接选项部件』

相关任务

第 285 页的『编辑链接选项部件的 callLink 元素』

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

相关参考

第 370 页的『callLink 元素』

第 379 页的『callLink 元素中的 pgmName』

第 385 页的『callLink 元素中的 type』

callLink 元素中的 parmForm

链接选项部件的 callLink 元素的属性 **parmForm** 指定调用参数的格式。

如果要生成 Java 程序，则 **parmForm** 适用于以下情况：

- 属性 **pgmName** 是指由正在生成的程序调用的基于 CICS 的程序；并且
- 属性 **type** 为 `ejbCall` 或 `remoteCall`；在任何一种情况下，有效的 **parmForm** 值（如后面所述）都是 `COMMDATA`（缺省值）和 `COMMPTR`。

如果要生成 Java 包装器，则 **parmForm** 适用于以下情况：

- 属性 **pgmName** 是指通过 Java 包装器调用的生成的 COBOL 程序；并且
- 属性 **type** 为 `ejbCall` 或 `remoteCall`；在任何一种情况下，有效的 **parmForm** 值（如后面所述）都是 `COMMDATA`（缺省值）或 `COMMPTR`。

从以下列表中选择一个值:

COMMDATA

指定调用程序将业务数据（而不是指向数据的指针）放在 COMMAREA 中。

将每个自变量值移动到缓冲区中，并与上一个值相邻，而不考虑边界对齐。

如果属性 **type** 为 `ejbCall` 或 `remoteCall`，则 COMMDATA 是缺省值。

COMMPTR

指定调用程序执行下列操作:

- 将一系列 4 字节指针放在 COMMAREA 中，对传递的每个自变量放置一个指针
- 将最后一个指针的高位设置为 1

如果属性 **type** 的值为 `localCall`，则 COMMPTR 是缺省值。

相关概念

第 282 页的『链接选项部件』

相关任务

第 285 页的『编辑链接选项部件的 `callLink` 元素』

相关参考

第 370 页的『`callLink` 元素』

第 375 页的『`callLink` 元素中的 `linkType`』

第 378 页的『`callLink` 元素中的 `parmForm`』

『`callLink` 元素中的 `pgmName`』

第 385 页的『`callLink` 元素中的 `type`』

callLink 元素中的 `pgmName`

链接选项部件的 `callLink` 元素的属性 **`pgmName`** 指定 `callLink` 元素引用的程序部件的名称。

在程序名中可以将星号 (*) 用作全局替换字符；但是，仅当该字符是最后一个字符时才有效。有关详细信息，请参阅[链接选项部件](#)。

相关概念

第 282 页的『链接选项部件』

相关任务

第 285 页的『编辑链接选项部件的 `callLink` 元素』

相关参考

第 370 页的『`callLink` 元素』

callLink 元素中的 `providerURL`

链接选项部件的 `callLink` 元素的属性 **`providerURL`** 指定一个名称服务器的主机名和端口号，EGL 生成的 Java 程序或包装器使用该名称服务器来定位 EJB 会话 bean，而 EJB 会话 bean 又调用 EGL 生成的 Java 程序。此属性必须具有以下格式:

```
iiop://hostName:portNumber
```

hostName

名称服务器运行所在的机器的 IP 地址或主机名

portNumber

名称服务器所侦听的端口号

仅在下列情况下，属性 **providerURL** 才适用：

- 属性 **type** 的值为 `ejbCall`；并且
- 属性 **pgmName** 是指从正在生成的 Java 程序或包装器中调用的程序。

用双引号将 URL 引起来，以避免在端口号之前的句点或冒号出现问题。

如果未对 **providerURL** 指定值，则将使用缺省值。缺省值指示 EJB 客户机将查找位于本地主机上并且侦听端口 900 的名称服务器。缺省值等同于以下 URL：

```
"iop://"
```

以下 **providerURL** 值指示 EJB 客户机查找称为 `bankserver.mybank.com` 并且侦听端口 9019 的远程名称服务器：

```
"iop://bankserver.mybank.com:9019"
```

以下属性值指示 EJB 客户机查找称为 `bankserver.mybank.com` 并且侦听端口 900 的远程名称服务器：

```
"iop://bankserver.mybank.com"
```

相关概念

第 282 页的『链接选项部件』

相关任务

第 285 页的『编辑链接选项部件的 `callLink` 元素』

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

相关参考

第 370 页的『`callLink` 元素』

第 379 页的『`callLink` 元素中的 `pgmName`』

第 385 页的『`callLink` 元素中的 `type`』

callLink 元素中的 refreshScreen

链接选项部件的 `callLink` 元素的 **refreshScreen** 属性指示当被调用程序返回控制权时屏幕是否自动刷新。有效值为 `yes`（缺省值）和 `no`。

如果调用程序位于将文本表单显示到屏幕的运行单元中，并且存在下列任何一种情况，则将 **refreshScreen** 设置为 `no`：

- 被调用程序不显示文本表单；或者
- 调用程序在调用之后写全屏文本表单。

属性 **refreshScreen** 仅在下列情况下才适用：

- `callLink type` 属性是 `localCall`；或者
- 当 `remoteComType` 属性是 `direct` 或 `distinct` 时，`callLink type` 属性是 `remoteCall`。

如果将 **noRefresh** 指示符包括在 `call` 语句中，则此属性被忽略。

相关参考

第 513 页的『call』

callLink 元素中的 remoteBind

链接选项部件的 callLink 元素的属性 **remoteBind** 指定链接选项是在生成时还是在运行时生成的。仅在以下情况下，此属性才适用：

- 属性 **type** 的值为 `ejbCall` 或 `remoteCall`；并且
- 正在生成 Java 程序或包装器。属性 **pgmName** 可能是指由正在生成的程序调用的程序，在这种情况下，该条目是指从程序到程序的调用。另外，该属性也可能是指正在生成的程序，在这种情况下，该条目是指从包装器到程序的调用。

选择下列其中一个值：

GENERATION

在生成时指定的链接选项在运行时是必需的。GENERATION 是缺省值。

RUNTIME

在生成时指定的链接选项可以在部署时进行修正。在这种情况下，必须在运行时环境中包括链接属性文件。

EGL 在下列情况下将生成链接属性文件：

- 正在生成 Java 程序或包装器；
- 您将属性 **remoteBind** 设置为 `RUNTIME`；并且
- 您在构建描述符选项 **genProperties** 设置为 `GLOBAL` 或 `PROGRAM` 的情况下生成。

相关概念

第 282 页的『链接选项部件』

第 330 页的『链接属性文件』

相关任务

第 329 页的『部署链接属性文件』

第 285 页的『编辑链接选项部件的 callLink 元素』

相关参考

第 370 页的『callLink 元素』

第 358 页的『genProperties』

第 598 页的『链接属性文件（详细信息）』

第 379 页的『callLink 元素中的 pgmName』

第 385 页的『callLink 元素中的 type』

callLink 元素中的 remoteComType

链接选项部件的 callLink 元素的属性 **remoteComType** 指定在以下情况下使用的通信协议：

- 属性 **type** 的值为 `ejbCall` 或 `remoteCall`；并且
- 要生成 Java 程序或包装器：
 - 如果要生成 Java 程序，则属性 **pgmName** 是指由正在生成的程序调用的程序
 - 如果要生成 Java 包装器，则 **pgmName** 是指通过 Java 包装器调用的程序

选择下列其中一个值。

DEBUG

致使被调用程序在 EGL 调试器中运行，即使调用程序正在 Java 运行时或 Java 调试环境中运行亦如此。在下列情况下，您可能会使用此设置：

- 正在运行一个 Java 程序，该程序使用 EGL Java 包装器来调用通过 EGL 编写的程序；或者
- 正在运行 EGL 生成的调用程序，该程序调用通过 EGL 编写的程序。

上述情况可能发生在 WebSphere 测试环境之外，但也可能发生在该环境之内，当 JSP 调用通过 EGL 编写的程序时就是这种情况。在任何一种情况下，效果都是调用 EGL 源，而不是调用 EGL 生成的程序。

如果您正在使用 WebSphere 测试环境，则调用程序和被调用程序都必须在那里运行；不能从远程机器进行调用。

使用 DEBUG 时，在同一个 **callLink** 元素中设置下列属性：

- **library**，它命名包含被调用程序的项目
- **package**，它标识包含被调用程序的包；但是，如果调用程序和被调用程序位于同一个包中，则不需要设置此属性

如果调用程序不在 EGL 调试器中运行，并且不在 WebSphere 测试环境中运行，则必须设置 **callLink** 元素的下列属性：

- **serverid**，它应该指定侦听器的端口号（如果该端口号不是 8346 的话）；以及
- **location**，它必须包含正在运行 Eclipse 工作台的主机的名称。

DIRECT

指定调用程序或包装器使用直接本地调用，这意味着调用代码和被调用代码在同一个线程中运行。不涉及 TCP/IP 侦听器，并且属性 **location** 的值被忽略。DIRECT 是缺省值。

Java 调用程序不使用 EGL 中间件，但是，调用包装器使用该中间件来处理 EGL 与 Java 基本类型之间的数据转换。

如果 EGL 生成的 Java 代码要调用非 EGL 生成的动态链接库（DLL）或 C 或 C++ 程序，建议使用 **remoteComType** 值 DISTINCT。

DISTINCT

指定当以本地方式调用程序时启动新的运行单元。由于涉及 EGL 中间件，所以该调用仍被认为是远程的。

对于调用动态链接库（DLL）或 C 或 C++ 程序的 EGL 生成的 Java 程序，可以使用此值。

CICSECI

指定当您正在调试或运行访问 CICS 的非 J2EE 代码时，需要使用 CICS 事务网关（CTG）ECI 接口。

使用 CTG Java 类来实现此协议。要指定 CTG 服务器的 URL 和端口，请对 **callLink** 元素的 **ctgLocation** 和 **ctgPort** 属性赋值。要标识被调用程序所驻留的 CICS 区域，请指定 **location** 属性。

CICSJ2C

指定将 J2C 连接器用于 CICS 事务网关。

CICSSSL

指定使用 CICS 事务网关 (CTG) 的安全套接字层 (SSL) 功能。支持 SSL 的 JSSE 实现。

使用 CTG Java 类来实现此协议。要指定 CTG 服务器的其它信息, 请对下列 callLink 元素属性赋值:

- ctgKeyStore
- ctgKeyStorePassword
- ctgLocation
- ctgPort, 在这种情况下, 这是 CTG JSSE 侦听器侦听请求时所用的 TCP/IP 端口。如果未指定 ctgPort, 则使用 CTG 缺省端口 8050。

要标识被调用程序所驻留的 CICS 区域, 请指定 location 属性。

JAVA400

指定使用 IBM Toolbox for Java 来在 Java 包装器或程序与 (EGL 或 VisualAge Generator) 为 iSeries 生成的 COBOL 程序之间进行通信。

TCPIP

指定 EGL 中间件使用 TCP/IP。

相关概念

第 282 页的『链接选项部件』

相关任务

第 285 页的『编辑链接选项部件的 callLink 元素』

相关参考

第 373 页的『callLink 元素中的 ctgKeyStore』

第 373 页的『callLink 元素中的 ctgKeyStorePassword』

第 373 页的『callLink 元素中的 ctgLocation』

第 374 页的『callLink 元素中的 ctgPort』

第 285 页的『编辑链接选项部件的 callLink 元素』

第 325 页的『为 CICSJ2C 调用设置 J2EE 服务器』

第 326 页的『为 J2EE 应用程序客户机模块中的被调用程序设置 TCP/IP 侦听器』

第 320 页的『为调用的非 J2EE 应用程序设置 TCP/IP 侦听器』

callLink 元素中的 remotePgmType

链接选项部件的 callLink 元素的属性 **remotePgmType** 指定正在调用的程序的类型。此属性适用于以下情况:

- 属性 **type** 的值为 **ejbCall** 或 **remoteCall**; 并且
- 出现下列其中一种情况:
 - 如果要生成程序 (而不是包装器), 则属性 **pgmName** 是指由正在生成的程序调用的程序。

被调用程序为下列其中一种类型:

- EGL 生成的 Java 程序
- 非 EGL 生成的动态链接库 (DLL) 或 C 或 C++ 程序
- 在 CICS 上运行并且包含 CICS 命令的程序

- 如果要生成 Java 包装器，则 **pgmName** 是指通过 Java 包装器调用的程序。

EGL

被调用程序是由 EGL 或者由 VisualAge Generator 生成的 COBOL 或 Java 程序；在这种情况下，调用程序是 Java 程序或 Java 包装器。此值是缺省值。

相关概念

第 282 页的『链接选项部件』

第 8 页的『运行时配置』

相关任务

第 285 页的『编辑链接选项部件的 callLink 元素』

相关参考

第 370 页的『callLink 元素』

第 375 页的『callLink 元素中的 library』

第 379 页的『callLink 元素中的 pgmName』

第 385 页的『callLink 元素中的 type』

callLink 元素中的 serverID

链接选项部件的 callLink 元素的属性 **serverID** 指定下列其中一个值：

- 被调用程序的侦听器的 TCP/IP 端口号；但仅当使用 TCP/IP 协议时才指定此值。在这种情况下，不存在缺省值。
- 正被调用的 CICS 事务的标识，但仅当对 CICS 进行的访问是通过 CICS 事务网关的 ECI 接口或安全套接字层功能部件进行的时才指定此值。在这种情况下，缺省值是 CICS 服务器系统镜像事务。

仅在下列情况下才使用此属性：

- 属性 **type** 的值为 **ejbCall** 或 **remoteCall**；
- **remoteComType** 的值是 **TCPIP**、**CICSECI** 或 **CICSSSL**；并且
- 要生成 Java 程序或包装器：
 - 如果要生成 Java 程序，则属性 **pgmName** 是指由正在生成的程序调用的程序
 - 如果要生成 Java 包装器，则 **pgmName** 是指通过 Java 包装器调用的程序

相关概念

第 282 页的『链接选项部件』

相关任务

第 285 页的『编辑链接选项部件的 callLink 元素』

第 326 页的『为 J2EE 应用程序客户机模块中的被调用程序设置 TCP/IP 侦听器』

第 320 页的『为调用的非 J2EE 应用程序设置 TCP/IP 侦听器』

相关参考

第 370 页的『callLink 元素』

第 379 页的『callLink 元素中的 pgmName』

第 381 页的『callLink 元素中的 remoteComType』

第 385 页的『callLink 元素中的 type』

callLink 元素中的 type

链接选项部件的 `callLink` 元素的属性 `type` 指定调用类型。选择下列其中一个值：

ejbCall

指示生成的 Java 程序或包装器将通过使用 EJB 会话 bean 来实现程序调用，并且该 EJB 会话 bean 将访问属性 `pgmName` 中标识的程序。在下列任何一种情况下，值 `ejbCall` 都适用：

- 正在生成 Java 包装器，并且该包装器通过 EJB 会话 bean 来调用该程序。在这种情况下，属性 `pgmName` 是指从包装器调用的程序，并且使用 `ejbCall` 将导致生成 EJB 会话 bean。
- 正在生成 Java 程序，该程序通过 EJB 会话 bean 来调用生成的程序。在这种情况下，属性 `pgmName` 是指被调用程序，并且不生成 EJB 会话 bean。

在任何一种情况下，如果正在使用 EJB 会话 bean，则只要需要生成 EJB 会话 bean，就必须生成 Java 包装器。

必须将生成的会话 bean 部署到企业 Java 服务器上，并且下列其中一种情况必须成立：

- 用来查找 EJB 会话 bean 的名称服务器与调用该会话 bean 的代码位于同一机器上；或者
- 属性 `providerURL` 标识名称服务器所在的位置。

如果您希望使用 EJB 会话 bean，则必须使用链接选项部件来生成调用程序或包装器，在该链接选项部件中，被调用程序的 `type` 属性的值为 `ejbCall`。在部署时不能够作出使用会话 bean 的决定。但是，如果将属性 `remoteBind` 设置为 `RUNTIME`，则可以在部署时决定 EJB 会话 bean 如何访问生成程序，尽管在生成时作出此决定更有效。

localCall

指定调用不使用 EGL 中间件。在这种情况下，被调用程序与调用程序在同一进程中。

`localCall` 是缺省值。

remoteCall

指定调用使用 EGL 中间件，它将 12 个字节添加至传递的数据的结尾。这些字节允许调用程序从被调用程序接收返回值。

如果调用程序是 Java 代码，则通信是由属性 `remoteComType` 中指定的协议处理的；协议选项指示被调用程序是在同一线程中还是另一个线程中。

如果进行调用时传递了变长记录，则下列描述是适用的：

- 为该记录保留空间，并且空间的大小就是对该记录指定的最大长度
- 如果 `callLink` 属性 `type` 的值为 `remoteCall` 或 `ejbCall`，则变长项（如果有的话）必须位于记录内

相关概念

第 282 页的『链接选项部件』

相关任务

第 285 页的『编辑链接选项部件的 `callLink` 元素』

相关参考

- 第 370 页的『callLink 元素』
- 第 375 页的『callLink 元素中的 linkType』
- 第 376 页的『callLink 元素中的 location』
- 第 378 页的『callLink 元素中的 parmForm』
- 第 379 页的『callLink 元素中的 pgmName』
- 第 379 页的『callLink 元素中的 providerURL』
- 第 381 页的『callLink 元素中的 remoteComType』

将 C 函数与 EGL 配合使用

EGL 程序可调用 C 函数。

要从 EGL 调用 C 函数:

在标识要在 EGL 程序中使用的 C 函数之后, 必须:

1. 将 EGL 堆栈库和应用程序对象文件从 IBM Web 站点下载至您的计算机。
2. 将所有 C 代码编译到一个共享库中并将其链接至相应的特定于平台的堆栈库。
3. 创建函数表。
4. 将函数表和相应的特定于平台的应用程序对象文件编译到共享库中, 并将此共享库链接至在步骤 2 中创建的共享库和堆栈库。

1. 下载 EGL 堆栈库和应用程序对象文件

要下载 EGL 堆栈库和应用程序对象文件:

1. 找到 EGL Support Web 站点。
 - Rational Application Developer 的 URL 为:
<http://www3.software.ibm.com/ibmdl/pub/software/rationalsdp/rad/60/redirect>
 - Rational Web Developer 的 URL 为:
<http://www3.software.ibm.com/ibmdl/pub/software/rationalsdp/rwd/60/redirect>
2. 将 **EGLRuntimesV60IFix001.zip** 文件下载至您喜欢的目录。
3. 解压缩 **EGLRuntimesV60IFix001.zip** 以标识下列文件:

对于特定于平台的堆栈库:

- AIX: EGLRuntimes/Aix/bin/libstack.so
- Linux: EGLRuntimes/Linux/bin/libstack.so
- Win32:
EGLRuntimes/Win32/bin/stack.dll
EGLRuntimes/Win32/bin/stack.lib

对于特定于平台的应用程序对象文件:

- AIX: EGLRuntimes/Aix/bin/application.o
- Linux: EGLRuntimes/Linux/bin/application.o

- Win32: EGLRuntimes/Win32/bin/application.obj

2. 将所有 C 代码编译到共享库中

C 代码使用出栈外部函数从 EGL 接收值并使用返回外部函数将值返回至 EGL。出栈外部函数在从 EGL 接收值中作了描述；返回外部函数在将值返回至 EGL 中作了描述。

要将所有 C 代码编译到共享库中：

1. 使用标准方法，将所有 C 代码编译到一个共享库中并将其链接至相应的特定于平台的 EGL 堆栈库。
2. 在下列特定于平台的示例中，**file1.c** 和 **file2.c** 是包含从 EGL 调用的函数的 C 文件。

在 AIX 上 (ld 命令必须在同一行上)：

```
cc -c -Iincl_dir file1.c file2.c
ld -G -b32 -bexpall -bnoentry
  -brtl file1.o file2.o -Lstack_lib_dir
  -lstack -o lib1_name -lc
```

在 Linux 上 (gcc 命令必须在同一行上)：

```
cc -c -Iincl_dir file1.c file2.c
gcc -shared file1.o file2.o -Lstack_lib_dir
  -lstack -o lib1_name
```

在 Windows 上 (link 命令必须在同一行上)：

```
cl /c -Iincl_dir file1.c file2.c
link /DLL file1.obj file2.obj
  /LIBPATH:stack_lib_dir
  /DEFAULTLIB:stack.lib /OUT:lib1_name
```

incl_dir

头文件的目录位置。

stack_lib_dir

堆栈库的目录位置。

lib1_name

输出库的名称。

注：如果 C 代码要使用任何 IBM Informix ESQL/C 库函数 (BIGINT、DECIMAL、DATE、INTERVAL、DATETIME)，则还必须链接 ESQL/C 库。

3. 创建函数表

函数表是一个 C 源文件，该源文件包括要从 EGL 程序调用的所有 C 函数的名称。在以下函数表示例中，**c_fun1** 和 **c_fun2** 是 C 函数的名称。在代码中标识的所有函数必须已经从上面步骤 2 中创建的 C 共享库中导出。

```
#include <stdio.h>
struct func_table {
    char *fun_name;
    int (*fptr)(int);
};
```

```

extern int c_fun1(int);
extern int c_fun2(int);
/* Similar prototypes for other functions */

struct func_table ftab[] =
{
    "c_fun1", c_fun1,
    "c_fun2", c_fun2,
    /* Similarly for other functions */
    "", NULL
};

```

根据上述示例创建函数表，并使用相应的 C 函数填充函数表。用 "" 和 NULL 来指示函数表的结束。

4. 将函数表和特定于平台的应用程序对象文件编译到共享库中

应用程序对象文件在 EGL 代码与 C 代码之间的接口中。

下面两个构件必须编译到一个共享库中并且链接至堆栈库和上面步骤 2 中创建的库：

- 函数表
- 应用程序对象文件

使用以下示例编译新的共享库，其中 **ftable.c** 是函数表的名称，**mylib** 是步骤 2 中创建的 C 共享库的名称，而 **lib_dir** 是 **mylib** 的目录位置。通过使用 *dllName* 属性或 *vj.defaultI4GLNativeLibrary* Java 运行时属性来指定 **lib2_name**。

在 AIX 上 (ld 命令必须在同一行上)：

```

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

```

在 Linux 上 (gcc 命令必须在同一行上)：

```

cc -c ftable.c
gcc -shared ftable.o application.o
    -Lstack_lib_dir -lstack -Llib_dir
    -lmylib -o lib2_name

```

在 Windows 上 (link 命令必须在同一行上)：

```

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

```

将三个库链接在一起。

使用 C 共享库、函数表和链接的堆栈库，您就可以从 EGL 代码调用 C 函数了。有关如何在 EGL 中调用 C 函数的信息，请参阅从 *EGL 程序调用 C 函数*。

相关概念

第 282 页的『链接选项部件』

相关参考

- 『C 的 BIGINT 函数』
- 第 390 页的『C 数据类型和 EGL 基本类型』
- 第 391 页的『C 的 DATE 函数』
- 第 391 页的『C 的 DATETIME 和 INTERVAL 函数』
- 第 392 页的『C 的 DECIMAL 函数』
- 第 393 页的『从 EGL 程序调用 C 函数』
- 第 397 页的『返回 C 的函数』
- 第 395 页的『C 的堆栈函数』

C 的 BIGINT 函数

注：以下 BIGINT 函数仅对 IBM Informix ESQL/C 的用户可用。要使用这些函数，ESQL/C 用户需要手工将 C 代码链接至 ESQL/C 库。

BIGINT 数据类型是独立于机器的方法，用于表示范围从 $-2^{63}-1$ 到 $2^{63}-1$ 的数字。ESQL/C 提供了一些例程，可以很方便地将 BIGINT 数据类型转换为 C 语言中的其它数据类型。

BIGINT 数据类型在内部由 **ifx_int8_t** 结构表示。有关结构的信息可在头文件 **int8.h** 中找到，该头文件包括在 ESQL/C 产品中。将此文件加入使用任何 BIGINT 函数的所有 C 源文件中。

针对 **int8** 类型数字的所有操作必须使用对应 **int8** 数据类型的下列 ESQL/C 库函数来执行。任何其它操作、修改或分析可能会产生不可预测的结果。ESQL/C 库提供下列函数，它们允许您处理 **int8** 数字并将 **int8** 类型的数字转换为其它数据类型或从其它数据类型转换为 **int8** 类型。

函数名	描述
<code>ifx_int8add()</code>	加上两个 BIGINT 类型值
<code>ifx_int8cmp()</code>	比较两个 BIGINT 类型数字
<code>ifx_int8copy()</code>	复制 ifx_int8_t 结构
<code>ifx_int8cvasc()</code>	将 C char 类型值转换为 BIGINT 类型数字
<code>ifx_int8cvdbl()</code>	将 C double 类型数字转换为 BIGINT 类型数字
<code>ifx_int8cvdec()</code>	将 decimal 类型值转换为 BIGINT 类型值
<code>ifx_int8cvflt()</code>	将 C float 类型值转换为 BIGINT 类型值
<code>ifx_int8cvint()</code>	将 C int 类型数字转换为 BIGINT 类型数字
<code>ifx_int8cvlong()</code>	将 C long （在 64 位机器上为 int ）类型值转换为 BIGINT 类型值
<code>ifx_int8cvlong_long()</code>	将 C long long 类型（8 字节值，在 32 位系统上为 long long ，在 64 位系统上为 long ）值转换为 BIGINT 类型值
<code>ifx_int8div()</code>	除以两个 BIGINT 数字
<code>ifx_int8mul()</code>	乘以两个 BIGINT 数字
<code>ifx_int8sub()</code>	减去两个 BIGINT 数字
<code>ifx_int8toasc()</code>	将 BIGINT 类型值转换为 C char 类型值
<code>ifx_int8todbl()</code>	将 BIGINT 类型值转换为 C double 类型值
<code>ifx_int8todec()</code>	将 BIGINT 类型数字转换为 decimal 类型数字

函数名	描述
ifx_int8toflt()	将 BIGINT 类型数字转换为 C float 类型数字
ifx_int8toint()	将 BIGINT 类型值转换为 C int 类型值
ifx_int8tolong()	将 BIGINT 类型值转换为 C long (在 64 位机器上为 int) 类型值
ifx_int8tolong_long()	将 C long long (在 64 位机器上为 long) 类型值转换为 BIGINT 类型值

相关参考有关各个函数的更多信息，请参阅以下

IBM Informix ESQL/C Programmer's Manual。

第 391 页的『C 的 DATE 函数』

第 391 页的『C 的 DATETIME 和 INTERVAL 函数』

第 392 页的『C 的 DECIMAL 函数』

第 393 页的『从 EGL 程序调用 C 函数』

C 数据类型和 EGL 基本类型

下表显示 C 数据类型、I4GL 数据类型与 EGL 基本类型之间的映射。

C 数据类型	等效的 I4GL 数据类型	等效的 EGL 基本类型
char	CHAR 或 CHARACTER	UNICODE(1)
char	NCHAR	UNICODE(size)
char	NVARCHAR	STRING
char	VARCHAR	STRING
int	INT 或 INTEGER	INT
short	SMALLINT	SMALLINT
ifx_int8_t	BIGINT	BIGINT
dec_t	DEC 或 DECIMAL(p,s) 或 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

相关参考

第 46 页的『BIN 和整数类型』

第 45 页的『BLOB』

第 44 页的『CLOB』

第 38 页的『DATE』

- 第 46 页的『DECIMAL』
- 第 47 页的『FLOAT』
- 第 38 页的『INTERVAL』
- 第 393 页的『从 EGL 程序调用 C 函数』
- 第 36 页的『MBCHAR』
- 第 47 页的『MONEY』
- 第 47 页的『NUM』
- 第 31 页的『基本类型』
- 第 49 页的『SMALLFLOAT』
- 第 40 页的『TIME』
- 第 40 页的『TIMESTAMP』

C 的 DATE 函数

注：以下 DATE 函数仅对 IBM Informix ESQL/C 的用户可用。要使用这些函数，ESQL/C 用户需要手工将 C 代码链接至 ESQL/C 库。

下列日期处理函数在 ESQL/C 库中。它们在字符串格式与内部 DATE 格式之间转换日期。

函数名	描述
rdatestr()	将内部 DATE 转换为字符串格式
rdayofweek()	返回某个日期表示的星期几，以内部格式表示
rdefmtdate()	将指定字符串格式转换为内部 DATE
rfmtdate()	将内部 DATE 转换为指定字符串格式
rjulmdy()	从指定 DATE 返回月份、日期和年份
rleapyear()	确定指定年份是否为闰年
rmidyjul()	从月份、日期和年份返回内部 DATE
rstrdate()	从字符串格式转换为内部 DATE
rtoday()	将系统日期作为内部 DATE 返回

相关参考

有关各个函数的更多信息，请参阅以下 IBM Informix ESQL/C Programmer's Manual。

- 第 389 页的『C 的 BIGINT 函数』
- 『C 的 DATETIME 和 INTERVAL 函数』
- 第 392 页的『C 的 DECIMAL 函数』
- 第 393 页的『从 EGL 程序调用 C 函数』

C 的 DATETIME 和 INTERVAL 函数

注：下列 DATETIME 和 INTERVAL 函数仅对 IBM Informix ESQL/C 的用户可用。要使用这些函数，ESQL/C 用户需要手工将 C 代码链接至 ESQL/C 库。

DATETIME 和 INTERVAL 数据类型是用 `dttime_t` 和 `intrvl_t` 结构在内部分别表示的。有关这些结构的信息可在头文件 `datetime.h` 中找到，该头文件包括在 ESQL/C 产品中。将此文件加入使用任何 DATETIME 和 INTERVAL 函数的所有 C 源文件中。

必须将下列 ESQL/C 库函数用于 `datetime` 和 `interval` 数据类型以对这些类型的值执行所有操作。

函数名	描述
<code>dtaddinv()</code>	将时间间隔值加上日期时间值
<code>dtcurrent()</code>	获取当前日期和时间
<code>dtcvasc()</code>	将符合 ANSI 的字符串转换为日期时间值
<code>dtcvfmtasc()</code>	将带有指定格式的字符串转换为日期时间值
<code>dtextend()</code>	更改日期时间值的限定符
<code>dtsub()</code>	将一个日期时间值减去另一个日期时间值
<code>dsubinv()</code>	将日期时间值减去时间间隔值
<code>dttoasc()</code>	将日期时间值转换为符合 ANSI 的字符串
<code>dttofmtasc()</code>	将日期时间值转换为带有指定格式的字符串
<code>incvasc()</code>	将符合 ANSI 的字符串转换为时间间隔值
<code>incvfmtasc()</code>	将带有指定格式的字符串转换为时间间隔值
<code>intoasc()</code>	将时间间隔值转换为符合 ANSI 的字符串
<code>intofmtasc()</code>	将时间间隔值转换为带有指定格式的字符串
<code>invdivdbl()</code>	用时间间隔值除以数字值
<code>invdivinv()</code>	用一个时间间隔值除以另一个时间间隔值
<code>invextend()</code>	将时间间隔值延长至不同时间间隔限定符
<code>invmuldbl()</code>	用时间间隔值乘以数字值

相关参考

有关各个函数的更多信息，请参阅以下

IBM Informix ESQL/C Programmer's Manual。

第 389 页的『C 的 BIGINT 函数』

第 391 页的『C 的 DATE 函数』

『C 的 DECIMAL 函数』

第 393 页的『从 EGL 程序调用 C 函数』

C 的 DECIMAL 函数

注：以下 DECIMAL 函数仅对 IBM Informix ESQL/C 的用户可用。要使用这些函数，ESQL/C 用户需要手工将 C 代码链接至 ESQL/C 库。

数据类型 DECIMAL 是独立于机器的方法，用于表示最多 32 个有效数字位的数字，可以带或不带小数点，并且指数范围在 -128 与 +126 之间。ESQL/C 提供了一些例程，可以很方便地将 DECIMAL 类型的数字转换成 C 语言中允许的每种数据类型或从该数据类型转换成 DECIMAL 类型的数字。DECIMAL 类型的数字由基数为 100 的指数和尾数（或小数部分）组成。在标准化表中，尾数的第一位必须大于零。

DECIMAL 数据类型在内部由 **dec_t** 结构表示。**decimal** 结构和类型定义 **dec_t** 可在头文件 **decimal.h** 中找到，该头文件包括在 ESQL/C 产品中。将此文件加入使用任何十进制函数的所有 C 源文件中。

针对 **decimal** 类型数字的所有操作必须使用对应 **decimal** 数据类型的下列 ESQL/C 库函数来执行。任何其它操作、修改或分析可能会产生不可预测的结果。

函数名	描述
deccvasc()	将 C int1 类型转换为 DECIMAL 类型
dectoasc()	将 DECIMAL 类型转换为 C int1 类型
deccvint()	将 C int 类型转换为 DECIMAL 类型
dectoint()	将 DECIMAL 类型转换为 C int 类型
deccvlong()	将 C int4 类型转换为 DECIMAL 类型
dectolong()	将 DECIMAL 类型转换为 C int4 类型
deccvflt()	将 C float 类型转换为 DECIMAL 类型
dectoflt()	将 DECIMAL 类型转换为 C float 类型
deccvdbl()	将 C double 类型转换为 DECIMAL 类型
dectodbl()	将 DECIMAL 类型转换为 C double 类型
decadd()	加上两个 DECIMAL 数字
decsub()	减去两个 DECIMAL 数字
decmul()	乘以两个 DECIMAL 数字
decdiv()	除以两个 DECIMAL 数字
deccmp()	比较两个 DECIMAL 数字
deccopy()	复制 DECIMAL 数字
dececv()	将 DECIMAL 值转换为 ASCII 字符串
decfcvt()	将 DECIMAL 值转换为 ASCII 字符串

相关参考

有关各个函数的更多信息，请参阅以下

IBM Informix ESQL/C Programmer's Manual。

第 389 页的『C 的 BIGINT 函数』

第 391 页的『C 的 DATE 函数』

第 391 页的『C 的 DATETIME 和 INTERVAL 函数』

『从 EGL 程序调用 C 函数』

从 EGL 程序调用 C 函数

可从 EGL 程序调用 C 函数。在遵循下列指示信息之前，必须按将 C 函数与 EGL 配合使用中标识的那样编译并链接 C 代码。

要从 EGL 程序调用 C 函数：

1. 使用函数调用语句，指定下列各项：
 - C 函数的名称
 - 要传递至 C 函数的所有自变量
 - 要返回至 EGL 程序的所有变量

2. 创建包含函数定义的 EGL 本机库部件。
3. 使用 USE 语句，在调用模块中指定 EGL 本机库部件。

例如，以下函数调用语句调用 C 函数 `sendmsg()`

```
sendmsg(chartype, 4, msg_status, return_code);
```

它将两个自变量（分别是 `chartype` 和 `4`）传递至该函数并期望有两个自变量被传递回来（分别是 `msg_status` 和 `return_code`）。这是通过按如下所示在本机库中定义函数来明确的：

```
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
```

传递的自变量是使用“in”参数指定的，并且要返回的自变量是使用“out”参数指定的。

callingConvention

使用自变量堆栈机制指定要在函数与调用代码之间传递的自变量。

dllName

指定此函数所在的 C 共享库。

注：还将使用 `vgj.defaultI4GLNativeLibrary` 系统属性来指定 C 共享库名。如果已经指定 `dllName` 和系统属性，将使用 `dllName`。有关 EGL `nativeLibrary` 的更多信息，请参阅类型为 `nativeLibrary` 的库部件帮助主题。

C 函数接收一个整数自变量，它指定自变量堆栈上推送多少个值（在此情况下为两个自变量）。这是要在 C 函数中弹出堆栈的值的数目。在将控制权交还给 EGL 程序之前，该函数还需要返回 `msg_status` 和 `return_code` 自变量的值。出栈外部函数在从 EGL 接收值中作了描述；返回外部函数在将值返回至 EGL 中作了描述。

C 函数不应假定它已经传递了正确数目的堆栈化值。C 函数应测试其整数自变量以了解有多少个 EGL 自变量已堆栈化。

以下示例显示刚好需要一个自变量的 C 函数：

```
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 */
}

```

该函数返回指定日期后的下一个工作日的日期。因为函数必须刚好接收一个自变量，所以函数会检查传递的自变量数目。如果函数接收不同数目的自变量，它会终止该程序（并带有标识消息）。

相关参考

- 第 389 页的『C 的 BIGINT 函数』
- 第 390 页的『C 数据类型和 EGL 基本类型』
- 第 130 页的『创建 EGL 库部件』
- 第 391 页的『C 的 DATE 函数』
- 第 391 页的『C 的 DATETIME 和 INTERVAL 函数』
- 第 392 页的『C 的 DECIMAL 函数』
- 第 473 页的『函数调用』
- 第 131 页的『类型为 basicLibrary 的库部件』
- 『C 的堆栈函数』
- 第 397 页的『返回 C 的函数』
- 第 386 页的『将 C 函数与 EGL 配合使用』

C 的堆栈函数

EGL 使用自变量堆栈来调用 C 函数，这是在函数与调用代码之间传递自变量的一种机制。EGL 调用函数将其自变量推进堆栈，而被调用的 C 函数将它们弹出堆栈以使用这些值。被调用函数将其返回值推进堆栈，而调用程序将它们弹出来以检索这些值。出栈和返回外部函数是随自变量堆栈库提供的。下面按每次从自变量堆栈弹出的值的数据类型描述出栈外部函数。返回外部函数在返回 C 的函数中作了描述。

注：出栈函数最初与 IBM Informix 4GL (I4GL) 配合使用；因此会将“4gl”包括在函数名中。

用于返回值的库函数

可从 C 函数调用下列库函数以将数字值弹出自变量堆栈：

- 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)

下表和下面类似的表映射 I4GL 版本 7.31 之前的版本与版本 7.31 和更新版本之间的返回函数名：

版本 7.31 之前的版本名称	版本 7.31 和更新版本名称
popint	ibm_lib4gl_popMInt
popshort	ibm_lib4gl_popInt2

版本 7.31 之前的版本名称	版本 7.31 和更新版本名称
poplong	ibm_lib4gl_popInt4
popflo	ibm_lib4gl_popFloat
popdub	ibm_lib4gl_popDouble
popdec	ibm_lib4gl_popDecimal

其中每个函数（如用于将值弹出堆栈的所有库函数）执行下列操作：

1. 从自变量堆栈除去一个值。
2. 在必要时转换其数据类型。如果堆栈上的值不能转换为指定的类型，将发生错误。
3. 将该值复制至指定的变量。

结构类型 **dec_t** 和 **ifx_int8_t** 用于表示 C 程序中的 DECIMAL 和 BIGINT 数据。有关用于处理和打印 DECIMAL 和 BIGINT 变量的 **dec_t** 和 **ifx_int8_t** 结构类型和库函数的更多信息，请参阅 *IBM Informix ESQL/C Programmer's Manual*。

用于将字符串弹出堆栈的库函数

可调用下列库函数以将字符串值弹出堆栈：

- 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)

版本 7.31 之前的版本名称	版本 7.31 和更新版本名称
popquote	ibm_lib4gl_popQuotedStr
popstring	ibm_lib4gl_popString
popvchar	ibm_lib4gl_popVarChar

ibm_lib4gl_popQuotedStr() 和 **ibm_lib4gl_popVarChar()** 将正好 **len** 个字节复制至字符串缓冲区 ***qv**。此处 **ibm_lib4gl_popQuotedStr()** 在必要时将填充空格，但 **ibm_lib4gl_popVarChar()** 不会填充至标准长度。复制至缓冲区的最后一个字节是用于终止字符串的空字节，所以最大字符串数据长度为 **len-1**。如果放入堆栈的自变量的长度超过 **len-1**，则结尾字节会丢失。

len 自变量设置接收字符串缓冲区的最大大小。使用 **ibm_lib4gl_popQuotedStr()**，可以刚好接收 **len** 个字节（包括结尾空格和空字节），即使堆栈上的值为空字符串也是如此。要查找 **ibm_lib4gl_popQuotedStr()** 检索到的字符串的真实数据长度，必须修剪出栈值的空格。

注：除了函数 **ibm_lib4gl_popString()** 会自动修剪所有结尾空格外，函数 **ibm_lib4gl_popString()** 和 **ibm_lib4gl_popQuotedStr()** 是完全相同的。

用于将时间值弹出堆栈的库函数

可调用下列库函数以将 DATE、INTERVAL 和 DATETIME (TIMESTAMP) 弹出堆栈：

- extern void ibm_lib4gl_popDate(int *datv)
- extern void ibm_lib4gl_popInterval(intrvl_t *iv, int qual)

可调用下列库函数以将 `TIMESTAMP` 值弹出堆栈:

- `extern void ibm_lib4gl_popDateTime(dtime_t *dtv, int qual)`

版本 7.31 之前的版本名称	版本 7.31 和更新版本名称
<code>popdate</code>	<code>ibm_lib4gl_popDate</code>
<code>popdtime</code>	<code>ibm_lib4gl_popDateTime</code>
<code>popinv</code>	<code>ibm_lib4gl_popInterval</code>

结构类型 `dtime_t` 和 `intrvl_t` 用于表示 C 程序中的 `DATETIME` 和 `INTERVAL` 数据。`qual` 自变量接收 `DATETIME` 或 `INTERVAL` 限定符的二进制表示。有关用于处理和打印 `DATE`、`DATETIME` 和 `INTERVAL` 变量的 `dtime_t` 和 `intrvl_t` 结构类型和库函数的更多信息, 请参阅 *IBM Informix ESQL/C Programmer's Manual*。

用于将 `BYTE` 或 `TEXT` 值弹出堆栈的库函数

可调用以下函数以将 `BYTE` 或 `TEXT` 自变量弹出堆栈:

- `extern void ibm_lib4gl_popBlobLocator(loc_t **blob)`

版本 7.31 之前的版本名称	版本 7.31 和更新版本名称
<code>poplocator</code>	<code>ibm_lib4gl_popBlobLocator</code>

结构类型 `loc_t` 定义 `BYTE` 或 `TEXT` 值, 这在 *IBM Informix ESQL/C Programmer's Manual* 中作了讨论。

`BYTE` 或 `TEXT` 自变量必须作为 `BYTE` 或 `TEXT` 弹出堆栈, 原因是 EGL 不提供自动数据类型转换。

相关参考

- 第 389 页的『C 的 `BIGINT` 函数』
- 第 390 页的『C 数据类型和 EGL 基本类型』
- 第 386 页的『将 C 函数与 EGL 配合使用』
- 第 391 页的『C 的 `DATE` 函数』
- 第 391 页的『C 的 `DATETIME` 和 `INTERVAL` 函数』
- 第 392 页的『C 的 `DECIMAL` 函数』
- 第 393 页的『从 EGL 程序调用 C 函数』
- IBM Informix ESQL/C Programmer's Manual*
『返回 C 的函数』

返回 C 的函数

EGL 使用自变量堆栈来调用 C 函数, 这是在函数与调用代码之间传递自变量的一种机制。EGL 调用函数将其自变量推进堆栈, 而被调用的 C 函数将它们弹出堆栈以使用这些值。被调用函数将其返回值推进堆栈, 而调用程序将它们弹出来以检索这些值。出栈外部函数和返回外部函数是随自变量堆栈库一起提供的。返回外部函数在下面作出描述; 使用的出栈外部函数在 C 的堆栈函数中作了描述。

外部返回函数将其自变量复制至在调用函数外分配的存储空间。此存储空间会在返回值被弹出堆栈时释放。这样就可以从函数的局部变量返回值了。

注：返回函数最初与 IBM Informix 4GL (I4GL) 配合使用；因此会将“4gl”包括在函数名中。

用于返回值的库函数

下列库函数对返回值可用：

- 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)

下表映射 I4GL 版本 7.31 之前的版本与版本 7.31 和更新版本之间的返回函数名：

版本 7.31 之前的版本名称	版本 7.31 和更新版本名称
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
retstring	ibm_lib4gl_returnString
retvchar	ibm_lib4gl_returnVarChar
retdate	ibm_lib4gl_returnDate
retdtime	ibm_lib4gl_returnDateTime
retinv	ibm_lib4gl_returnInterval

ibm_lib4gl_returnQuotedStr() 的自变量是以 null 结束的字符串。包括 **ibm_lib4gl_returnString()** 函数只是为了对称；它在内部调用 **ibm_lib4gl_returnQuotedStr()**。

只要方便，C 函数可以返回任何格式的数据。如果可以转换，EGL 会在将值弹出堆栈时对数据类型进行必要的转换。如果不能进行数据类型转换，将发生错误。

从 EGL 调用的 C 函数总是使用语句 **return(n)** 退出的，其中 *n* 是推送至堆栈的返回值的数目。不返回任何内容的函数必须用 **return(0)** 退出。

相关参考

- 第 389 页的『C 的 BIGINT 函数』
- 第 390 页的『C 数据类型和 EGL 基本类型』
- 第 393 页的『从 EGL 程序调用 C 函数』
- 第 386 页的『将 C 函数与 EGL 配合使用』
- 第 391 页的『C 的 DATE 函数』
- 第 391 页的『C 的 DATETIME 和 INTERVAL 函数』
- 第 392 页的『C 的 DECIMAL 函数』
- 第 395 页的『C 的堆栈函数』

注释

EGL 文件中的注释是通过下列任何一种方式创建的:

- 双右向斜杠 (//) 指示后续字符都是注释, 直至并包括行结束字符
- 单行或多行注释是这样定界的: 开始处是一个右向斜杠和一个星号 (/*), 结束处是一个星号和一个右向斜杠 (*/); 只要空格字符有效, 此注释格式就有效

可将注释放在可执行语句的内部或外部, 如以下示例所示:

```
/* the assignment e = f occurs if a == b or if c == d */
if (a == b          // one comparison
    || /* OR; another comparison */ c == d)
    e = f;
end
```

EGL 不支持嵌入式注释, 因此下列条目会导致错误:

```
/* this line starts a comment /* and
   this line ends the comment, */
   but this line is not inside a comment at all */
```

前两行中的注释包括了另外一个注释定界符 (/*)。仅当 EGL 尝试将第三行解释为源代码时才会出错。

下列情况有效:

```
a = b; /* this line starts a comment // and
        this line ends the comment */
```

最后一个示例中的双右向斜杠 (//) 本身是更大型的注释的一部分。

在符号 #sql{ 与 } 之间, 前面描述的 EGL 注释无效。下列描述是适用的:

- SQL 注释在一行的开头或者在空格之后以双连字符 (--) 开始, 直到该行的末尾
- 在字符串文字中, 不能使用注释。即使在下列上下文中, 该文字中的字符序列也被解释为文本:
 - prepare 语句
 - SQLRecord 类型的记录的 **defaultSelectCondition** 属性

相关概念

- 第 13 页的『EGL 项目、包和文件』

相关参考

- 第 448 页的『EGL 源格式』
- 第 80 页的『EGL 语句』

与 VisualAge Generator 的兼容性

EGL 是 VisualAge Generator 4.5 的替代，它包含一些主要用来使您能够将现有程序迁移至新开发环境的语法。在开发环境中，如果选择了 EGL 首选项 **VAGCompatibility** 或者（在生成或调试时）将构建描述符选项 **VAGCompatibility** 设置为 *yes*，则支持此语法。该首选项的设置还建立了该构建描述符选项的缺省值。

当 VisualAge Generator 兼容性起作用时，下列描述是适用的：

- 三个 otherwise 无效字符 (- @ #) 在标识中是有效的，尽管连字符 (-) 和磅符 (#) 在任何情况下作为第一个字符都是无效的；有关详细信息，请参阅命名约定
- 如果在不指定下标的情况下引用结构项的一维静态数组，则数组下标缺省为 1；有关详细信息，请参阅数组
- 可以使用基本类型 NUMC 和 PACF，如基本类型中所述
- 如果对具有基本类型 DECIMAL 的项指定偶数长度，则 EGL 将把长度增加 1，除非该项被用作 SQL 主变量。
- SQL 项属性 **SQLDataCode** 可用，如 SQL 项属性中所述
- 在 call 语句中，有一组调用选项可用
- 选项 **externallyDefined** 位于语句 show 和 transfer 中
- 下列系统变量可用：

- VGVar.handleSysLibraryErrors
- ConverseVar.segmentedMode

- 下列系统函数可用：

- VGLib.getVAGSysType
- VGLib.connectionService

- 可以发出具有以下格式的语句：

```
display printForm
```

```
printForm
```

对程序可视的打印表单的名称。

在该情况下，**display** 等同于 **print**。

- 以下程序属性在所有情况下可用，并且对于用 VisualAge Generator 编辑的代码特别有用：

- **allowUnqualifiedItemReferences**
- **handleHardIOErrors**（在设置为 *no* 时）
- **includeReferencedFunctions**
- **localSQLScope**（在设置为 *yes* 时）
- **throwNrfEofExceptions**（在设置为 *yes* 时）

有关详细信息，请参阅 *EGL 源格式的程序部件*。

- 如果设置文本表单属性 **value**，则仅当用户返回表单之后才能在程序中使用该属性的内容。因此，在程序中设置的值不需要对程序中的项有效。

要访问有关将 VisualAge Generator 程序迁移至 EGL 的完整详细信息，请参阅有关 *EGL* 的其它信息的来源。

相关概念

第 11 页的『有关 EGL 的其它信息的来源』

相关参考

第 68 页的『数组』

第 513 页的『call』

第 672 页的『输入表单』

第 672 页的『输入记录』

第 612 页的『命名约定』

第 625 页的『pfKeyEquate』

第 31 页的『基本类型』

第 576 页的『print』

第 664 页的『EGL 源格式的程序部件』

第 588 页的『show』

第 61 页的『SQL 项属性』

第 839 页的『connectionService()』

第 842 页的『getVAGSysType()』

第 869 页的『handleSysLibraryErrors』

第 848 页的『segmentedMode』

第 589 页的『transfer』

ConsoleUI

ConsoleField 属性和字段

下列属性在类型为 ConsoleField 的变量中是必需的:

- **fieldLen** (除非 ConsoleField 是常量字段)
- **position**

尽管 **name** 字段不在常量 ConsoleField 中, 但也是必需的。

ConsoleField 的属性如下所示:

fieldLen

指定显示有意义的最大值所需的位数。对于常量 consoleField, 不要设置此属性:

fieldLen 是显示值占用的字符数, **value** 属性中包括了这种设置。

类型: *INT*

示例: *fieldLen = 20*

缺省值: *none*

position

控制台字段在表单中的位置。该属性包含由两个整数组成的数组: 行号和跟在行号后面的列号。行号从表单顶部开始计算。同样, 列号从表单左边开始计算。

类型: *INT[]*

示例: *position = [2, 3]*

缺省值: *[1,1]*

segments

指定每个字段分段的行、列和长度，这是可以有定界符的 `consoleField` 子段。要创建多行文本框的外观，将一个字段分段放在同一表单列的连续行上，这样一组分段将充当一个字段。

类型: `INT[3][[]]`

示例: `segments = [[5,1,10],[6,1,10]]`

缺省值: `none`

如果对 **segments** 指定了值，则 **position** 的值将被忽略，**fieldLen** 应设置为所有组合分段的长度。

如果指定多个分段，则 `ConsoleField` 的行为还会受 **lineWrap** 字段的影响。

validValues

指定对用户输入有效的值列表。

类型: 单值元素和双值元素的数组文字

示例: `validValues = [[1,3], 5, 12]`

缺省值: `none`

有关详细信息，请参阅 `validValues`。

`consoleField` 数组的属性包括上面提到的属性 (**segments** 除外) 和下面的属性:

columns

指定在类型为 `ConsoleField` 的数组中显示元素的列数。例如，如果数组有 5 个元素并且 **columns** 属性的值为 2，则表单的第一行显示 2 个元素；第二行显示 2 个元素；第三行显示 1 个元素。

类型: `INT`

示例: `columns = 3`

缺省值: `1`

此属性仅对类型为 `ConsoleField` 的数组有意义。屏幕上的数组元素的分布（横向或纵向）受属性 **orientIndexAcross** 的影响。

linesBetweenRows

指定包含数组元素的每行之间的空白行数。

类型: `INT`

示例: `linesBetweenRows = 3`

缺省值: `0`

此属性仅对类型为 `ConsoleField` 的数组有意义。

orientIndexAcross

指示数组元素在屏幕上是否横向分布，如后面示例中所示。

类型: `Boolean`

示例: `orientIndexAcross = yes`

缺省值: `yes`

此属性仅对类型为 `consoleField` 的数组有意义。

如果属性 **orientIndexAcross** 设置为 `yes`，则数组的连续元素将以从左至右的方式显示。在下面的 2 列示例中，每个连续元素显示相当于元素下标的整数:

```
1 2
3 4
5
```

如果属性 **orientIndexAcross** 设置为 *no*，则连续元素将以从上至下的方式显示：

```
1 4
2 5
3
```

spacesBetweenColumns

指定将字段的每一列分开的空格数

类型: *INT*

示例: *spacesBetweenColumns = 3*

缺省值: *1*

此属性仅对类型为 `consoleField` 的数组有效。

`ConsoleField` 的字段如下所示：

align

当数据长度小于字段长度时，**align** 字段指定数据在变量字段中的位置。

类型: *AlignKind*

示例: *align = left*

缺省值: 对于字符或时间戳记数据为 *left*，对于数字为 *right*

在运行时是否可更新? 是

值如下所示：

left

将数据放在字段的左端。位于开头的空格将被除去，并且被放在字段末尾。

none

不对齐数据。此设置仅对字符数据有效。

right

将数据放在字段的右端。位于结尾的空格将被除去，并且被放在字段开头。

autonext

指示用户填写当前 `ConsoleField` 后光标是否转至下一字段。

类型: *Boolean*

示例: *autonext = yes*

缺省值: *None*

在运行时是否可更新? 是

跳进顺序确定哪个 `ConsoleField` 是下一个字段，如 *ConsoleUI 部件和相关变量* 中所述。

binding

指定缺省情况下与 `ConsoleField` 绑定的变量的名称。

类型: *String*

示例: *binding = "myVar"*

缺省值: *None*

在运行时是否可更新? 否。

有关绑定概述的信息，请参阅 *ConsoleUI* 部件和相关变量。

caseFormat

指定如何针对大小写区分问题处理输入和输出。

类型: *CaseFormatKind*

示例: *caseFormat = lowerCase*

缺省值: *defaultCase*

在运行时是否可更新? 是

值如下所示:

defaultCase (缺省值)

对大小写没有影响。

lowerCase

将字符尽可能变换为小写

upperCase

将字符尽可能变换为大写

color

指定 *ConsoleField* 中的文本颜色。

类型: *ColorKind*

示例: *color = red*

缺省值: *white*

在运行时是否可更新? 是, 但仅当在更新字段后显示 *ConsoleField* (或 *ConsoleField* 处于焦点位置) 时才会有更新的视觉效果

值如下所示:

defaultColor 或 **white** (缺省值)

白色

black

黑色

blue

蓝色

cyan

青色

green

绿色

magenta

品红色

red

红色

yellow

黄色

comment

指定注释, 这是光标在 *ConsoleField* 中时在特定于窗口的注释行 (如果有的话) 中显示的文本。

类型: *String*

示例: *"Employee name"*

缺省值: *Empty string*

在运行时是否可更新? 否

commentKey

指定用于搜索包括注释的资源束的键, 注释是光标在 `ConsoleField` 中时在特定于窗口的注释行 (如果有的话) 中显示的文本。如果同时指定 **comment** 和 **commentKey**, 则使用 **comment**。

类型: *String*

示例: *commentKey = "myKey"*

缺省值: *Empty string*

在运行时是否可更新? 否

资源束是由系统变量 **ConsoleLib.messageResource** 标识的, 如 *messageResource* 中所述。

dataType

指定用于标识数据类型的字符串。该值用于验证用户输入 (如 *= 1.5*) 是否与特定种类的 SQL 列兼容。仅当 `ConsoleField` (或相关 `ConsoleForm`) 的 **openUI** 语句包括语句属性 **isConstruct** 时, 此字段才有意义。

类型: *String*

示例: *dataType = "NUMBER"*

缺省值: *Empty string*

在运行时是否可更新? 否

对于数字输入, 如果允许用户指定浮点值 (在此情况下, *> 1.5* 是有效用户输入), 则指定值 *"NUMBER"*; 否则指定相当于整数的字符串; 如 *"INT"*。

dateFormat

指示如何定义输出的格式; 但仅当 `ConsoleField` 接受日期时才应指定 **dateFormat**。

类型: 字符串或与日期有关的系统常量

示例: *dateFormat = isoDateFormat*

缺省值: *none*

在运行时是否可更新? 否

有效值如下所示:

"pattern"

pattern 值由一组字符组成, 如日期、时间和时间戳记格式说明符中所述。

可从完整日期说明的开头或结尾删除字符, 但不能从该说明的中间删除字符。

defaultDateFormat

运行时 Java 语言环境中指定的日期格式。

isoDateFormat

模式“yyyy-MM-dd”, 这是国际标准组织 (ISO) 指定的日期格式。

usaDateFormat

模式“MM/dd/yyyy”, 这是 IBM 美国标准日期格式。

eurDateFormat

模式“dd.MM.yyyy”，这是 IBM 欧洲标准日期格式。

jisDateFormat

模式“yyyy-MM-dd”，这是日本工业标准日期格式。

systemGregorianCalendarFormat

8 个或 10 个字符的模式，它包括 dd（表示数字格式的当月某天）、MM（表示数字格式的月份）和 yy 或 yyyy（表示数字格式的年份），并将 d、M、y 或数字以外的字符用作分隔符。

该格式在以下 Java 运行时属性中：

```
vgj.datemask.gregorian.long.NLS
```

NLS

Java 运行时属性 **vgj.nls.code** 中指定的（本地语言支持）代码。此代码是 targetNLS 中列示的代码之一。不支持大写英语（代码 ENP）

有关 **vgj.nls.code** 的详细信息，请参阅 *Java 运行时属性（详细信息）*。

systemJulianDateFormat

6 个或 8 个字符的模式，它包含 DDD（表示数字格式的当年某天）和 yy 或 yyyy（表示数字格式的年份），并将 D、y 或数字以外的字符用作分隔符。

该格式在以下 Java 运行时属性中：

```
vgj.datemask.julian.long.NLS
```

NLS

Java 运行时属性 **vgj.nls.code** 中指定的（本地语言支持）代码。此代码是 targetNLS 中列示的代码之一。不支持大写英语（代码 ENP）

有关 **vgj.nls.code** 的详细信息，请参阅 *Java 运行时属性（详细信息）*。

editor

对用户与数据的交互指定该程序；但仅当 ConsoleField 与类型为 LOB 的变量绑定时有意义。

类型: *String*

示例: `editor = "/bin/vi"`

缺省值: *none*

在运行时是否可更新? 是

可指定 PATH 或 LIBPATH 中的可执行文件的名称；或者可指定该可执行文件的标准路径。

help

指定出现以下情况时要显示的文本：

- 光标在 ConsoleField 中；并且
- 用户按了 **ConsoleLib.key_help** 中标识的键。

类型: *String*

示例: `help = "Update the value"`

缺省值: *Empty string*

在运行时是否可更新? 是

helpKey

指定用于搜索资源束的访问键，该资源束包含在出现以下情况时将显示的文本：

- 光标在 `ConsoleField` 中；并且
- 用户按了 `ConsoleLib.key_help` 中标识的键。

如果同时指定 `help` 和 `helpKey`，则使用 `help`。

类型: *String*

示例: `helpKey = "myKey"`

缺省值: *Empty string*

在运行时是否可更新? 是

资源束是由系统变量 `ConsoleLib.messageResource` 标识的，如 `messageResource` 中所述。

highlight

指定显示 `ConsoleField` 时要使用的特殊效果（如果有的话）。

类型: *HighlightKind[]*

示例: `highlight = [reverse, underline]`

缺省值: *[noHighLight]*

在运行时是否可更新? 是，但仅当在更新 `highlight` 字段后显示 `ConsoleField`（或 `ConsoleField` 处于焦点位置）时才会有更新的视觉效果

值如下所示：

noHighlight（缺省值）

不会有特殊效果。使用此值将覆盖任何其它值。

blink

没有效果。

reverse

反转文本和背景色，这样的话（举例来说），如果显示器是黑底白字的，则背景变为白色的，而文本变为黑色的。

underline

在受影响区域下面加下划线。下划线的颜色就是文本的颜色，即使同时指定了值 `reverse` 也是如此。

initialValue

指定要显示的初始值。

类型: *String*

示例: `initialValue = "200"`

缺省值: *Empty string*

在运行时是否可更新? 是

如果 `openUI` 语句中的 `setInitial` 属性设置为 `true`，则将使用 `consoleField` 中的 `initialValue` 属性的值。但是，如果该 `openUI` 属性为 `false`，则改为显示绑定变量的当前值；并且 `initialValue` 属性的值会被忽略。

initialValueKey

指定用于搜索资源束的访问键，该资源束包含要显示的初始值。如果同时指定 `initialValue` 和 `initialValueKey`，则使用 `initialValue`。

类型: *String*

示例: `initialValueKey = "myKey"`

缺省值: *Empty string*

在运行时是否可更新? 是

资源束是由系统变量 **ConsoleLib.messageResource** 标识的, 如 *messageResource* 中所述。

inputRequired

指示是否用户不输入值就不能离开字段。

如果同时指定 **initialValue** 和 **initialValueKey**, 则使用 **initialValue**。

类型: *String*

示例: `initialValueKey = "myKey"`

缺省值: *Empty string*

在运行时是否可更新? 否

资源束是由系统变量 **ConsoleLib.messageResource** 标识的, 如 *messageResource* 中所述。

intensity

指定显示字体的强度。

类型: *IntensityKind[]*

示例: `intensity = [bold]`

缺省值: *[normalIntensity]*

在运行时是否可更新? 是, 但仅当在更新 **intensity** 字段后显示 *ConsoleField* (或 *ConsoleField* 处于焦点位置) 时才会有更新的视觉效果

值如下所示:

normalIntensity (缺省值)

不会有特殊效果。使用此值将覆盖任何其它值。

bold

使文本以粗体字体显示。

dim

现在此值不生效。将来, 输入字段被禁用或应该取消强调时, 可以适当地让文本以较低强度出现。

invisible

除去任何有关“字段位于表单中”的指示。

isBoolean

指示 *ConsoleField* 是否表示布尔值。字段 **isBoolean** 限制有效的 *ConsoleField* 值, 并且对输入或输出很有用。

数字字段的值是 0 (表示 *false*) 或 1 (表示 *true*)。

字符字段的值是由从属于本地语言的字或字的一部分表示的, 并且特定值由语言环境确定。例如, 在英语中, 长度为三个或更多个字符的布尔字段具有值 *yes* (表示 *true*) 或 *no* (表示 *false*), 一字符布尔字段值具有截断的值 *y* 或 *n*。

类型: *Boolean*

示例: *isBoolean = yes*

缺省值: *no*

在运行时是否可更新? 否

lineWrap

指示如何让文本在必要时换行以避免截断文本。

类型: *LineWrapType*

示例: *value = compress*

缺省值: *character*

在运行时是否可更新? 是

值如下所示:

character (缺省值)

字段中的文本不会在空格处断开, 但会在字段分段边界的字符位置断开。

compress

如果可能, 文本将在空格处断开。当用户离开 `ConsoleField` (通过浏览至另一 `ConsoleField` 或通过按 **Esc** 键) 时, 该值将赋给绑定变量, 并且将除去用来让文本换行的所有附加空格。

word

如果可能, 字段中的文本将在空格处断开。将该值赋给绑定变量时, 将包含附加空格以反映如何填充该值以便在字的边界换行。

lineWrap 字段只对具有多个分段的 `ConsoleField` 有意义, 因为它是由 **segments** 属性控制的。

masked

指示当用户输入密码时是否让 `ConsoleField` 中的每个字符相应地显示为星号 (*)。

类型: *Boolean*

示例: *masked = yes*

缺省值: *no*

在运行时是否可更新? 是

minimumInput

指示有效输入中的最小字符数。

类型: *INT*

示例: *minimumInput = 4*

缺省值: *no*

在运行时是否可更新? 否

name

`ConsoleField` 名称, 在运行时解析名称的编程环境中使用。强烈建议名称字段的值与变量名称相同。

类型: *String*

示例: *name = "myField"*

缺省值: *none*

在运行时是否可更新? 否

numericFormat

指示如何定义输出的格式；但仅当 `ConsoleField` 接受数字时才应指定 **numericFormat**。

类型: *String*

示例: `numericFormat = "-###@"`

缺省值: *none*

在运行时是否可更新? 否

有效字符如下所示:

- # 表示一位数字的占位符。
- * 将星号 (*) 用作前导零的填充字符。
- & 将零用作前导零的填充字符。
- # 将空格用作前导零的填充字符。
- < 使数字向左对齐。
- , 除非该位置包含前导零, 否则使用根据语言环境确定的数字分隔符。
- . 使用根据语言环境确定的小数点。
- 使用减号 (-) 来表示小于 0 的值; 使用空格来表示大于或等于 0 的值。
- + 使用减号 (-) 来表示小于 0 的值; 使用加号 (+) 来表示大于或等于 0 的值。
- (在记帐时, 适当地在负值前加上左圆括号。
-) 在记帐时, 适当地在负值后加上右圆括号。
- \$ 根据语言环境在该值前加上适当的货币符号。
- @ 根据语言环境在该值后加上适当的货币符号。

pattern

如果 `ConsoleField` 内容是字符类型, 则指定用于定义输入和输出格式的模式。

类型: *String*

示例: `pattern = "(###) ###-####"`

缺省值: *none*

在运行时是否可更新? 否

下列控制字符可用:

- A 是表示字母的占位符; 并且被视为字母的一部分字符是根据语言环境确定的
- # 是表示数字位的占位符
- X 是表示任意种类的必需字符的占位符

上面三种字符以外的字符包括在输入或输出中; 但对于输出, 所有被覆盖字符都将丢失:

- 如果输出模式为“(###) ###-####”, 则值“6219655561212”将为如下所示:
(219) 555-1212

原始值中的每个 6 将不会显示给用户, 并且会在数据存储更新时丢失。

- 对于输入，光标会跳过文字字符并且只允许在出现占位符的位置输入。在当前示例中，如果用户输入 2195551212，则字符串“(219) 555-1212”将变成 ConsoleField 中的值并且成为放在绑定变量中的值。

protect

指定是否保护 ConsoleField，不允许用户更新。

类型: *Boolean*

示例: *protect = yes*

缺省值: *no*

在运行时是否可更新? 否

值如下所示:

No (缺省值)

设置字段，使用户可以覆盖其中的值。

Yes

设置 consoleField，以使用户不能覆盖其中的值。另外，在下列情况下，每当用户尝试浏览至 consoleField 时光标将跳过该 consoleField:

- 用户正在依据跳进顺序在上一个 consoleField 中输入，并且按下 (a) **Tab** 键或者 (b) 在字段 **autonext** 设置为 *yes* 的情况下在上一个 consoleField 字段中填写内容。
- 用户正在依据跳进顺序在下一个 consoleField 中输入，并且按下 **Shift Tab** 键。
- 用户使用方向键移至上一个或下一个 consoleField。

可将变量与受保护或不受保护的 consoleField 绑定。openUI 属性 **setInitial** 的设置确定是否显示绑定变量的值。

如果程序尝试移至受保护的 consoleField，则会发生运行时错误。

SQLColumnName

指的是与 ConsoleField 相关联的数据库表列的名称。当 ConsoleField (或相关 ConsoleForm) 的 **openUI** 语句包括语句属性 **isConstruct** 时，此名称用于创建搜索条件。

类型: *String*

示例: *SQLColumnName = "ID"*

缺省值: *none*

在运行时是否可更新? 是

timeFormat

指示如何定义输出的格式; 但仅当 ConsoleField 接受时间时才应指定 **timeFormat**。

类型: *字符串或与时间有关的系统常量*

示例: *timeFormat = isoTimeFormat*

缺省值: *none*

在运行时是否可更新? 否

有效值如下所示:

"pattern"

pattern 值由一组字符组成，如日期、时间和时间戳记格式说明符中所述。

可从完整时间说明的开头或结尾删除字符，但不能从该说明的中间删除字符。

defaultTimeFormat

运行时 Java 语言环境中指定的时间格式。

isoTimeFormat

模式“HH.mm.ss”，这是国际标准组织（ISO）指定的时间格式。

usaTimeFormat

模式“hh:mm AM”，这是 IBM 美国标准时间格式。

eurTimeFormat

模式“HH.mm.ss”，这是 IBM 欧洲标准时间格式。

jisTimeFormat

模式“HH:mm:ss”，这是日本工业标准时间格式。

timestampFormat

指示如何定义输出的格式；但仅当 `ConsoleField` 接受时间戳记时才应指定 **timestampFormat**。

类型：字符串或与时间戳记有关的系统常量

示例：`timestampFormat = jdbcTimestampFormat`

缺省值：`none`

在运行时是否可更新？ 否

有效值如下所示：

"pattern"

pattern 值由一组字符组成，如日期、时间和时间戳记格式说明符中所述。

可从完整时间戳记说明的开头或结尾删除字符，但不能从该说明的中间删除字符。

defaultTimestampFormat

运行时 Java 语言环境中指定的时间戳记格式。

db2TimeStamFormat

模式“yyyy-MM-dd-HH.mm.ss.ffffff”，这是 IBM DB2 缺省时间戳记格式。

odbcTimestampFormat

模式“yyyy-MM-dd HH:mm:ss.ffffff”，这是 ODBC 时间戳记格式。

value

`consoleField` 中显示的当前值。代码可设置此值，以便调用 **ConsoleLib.displayForm** 时在 `consoleField` 中显示指定的值。

类型：`String`

示例：`value = "View"`

缺省值：`none`

在运行时是否可更新？ 是

verify

指示在尝试退出 `ConsoleField` 后是否提示用户重新输入相同的值。

类型: *String*

示例: `value = "View"`

缺省值: *none*

在运行时是否可更新? 是

值如下所示:

No (缺省值)

EGL 运行时不发出特殊提示。

Yes

当用户尝试离开 `ConsoleField` 时, EGL 运行时执行下列操作:

- 清除 `consoleField`, 让光标留在该处
- 显示一条消息, 指示用户重复输入内容
- 在用户再次尝试离开 `consoleField` 时比较两个输入值

如果两个值相匹配, 则绑定变量将接收该值并照常继续处理。如果两个值不匹配, 则 `consoleField` 内容会还原为两次用户输入中的第一次输入之前的值, 并且光标仍留在该字段中。

相关概念

第 163 页的『控制台用户界面』

相关参考

第 165 页的『ConsoleUI 部件和相关变量』

第 42 页的『日期、时间和时间戳记格式说明符』

第 493 页的『Java 运行时属性 (详细信息)』第 565 页的『openUI』

第 658 页的『validValues』

相关任务

第 164 页的『使用 consoleUI 创建界面』

EGL consoleUI 中的 ConsoleForm 属性

类型为 `ConsoleForm` 的记录部件的属性如下所示, 只有 `formSize` 是必需的:

delimiters

指定在输入字段之前和之后显示的字符。仅当属性 `showBrackets` 的值为 `yes` 时, 才显示这些字符。

类型: 字符串文字

示例: `delimiters = "<>/"`

缺省值: `"[]"`

只要可能, 就会在每个非常量 `ConsoleField` 之前显示第一个字符, 在每个非常量 `ConsoleField` 之后显示第二个字符。但是, 第三个字符显示在用两个位置隔开的两个非常量 `ConsoleField` 之间。

如果指定的字符不到三个, 则将对每个未指定字符使用缺省字符。如果指定的字符超过三个, 则第四个字符和后续字符将被忽略。

formSize

表单的维。该字段必须包含由两个正整数组成的数组: 行数和跟在行数后面的列数。

类型: *INT[2]*

示例: *size = [24, 80]*

缺省值: *none*

如果维超出显示表单的窗口大小, 则表单大小将会降低以装入到窗口维中。但是, 如果 `ConsoleField` 无法装入到窗口维中, 则程序将会终止。

name

表单名称, 在运行时解析名称的编程环境中使用。建议名称字段的值 (如果有的话) 与变量名称相同。

类型: *String*

示例: *name = "myForm"*

缺省值: *none*

名称字段用于系统函数, 如 **`ConsoleLib.displayFormByName`**。

showBrackets

指示是否用一对字符 (如方括号) 来对非常量 `ConsoleField` 定界。

类型: *Boolean*

示例: *showBrackets = no*

缺省值: *yes*

有关其它详细信息, 请参阅属性 **`delimiters`**。

相关概念

第 163 页的『控制台用户界面』

相关参考

第 165 页的『`ConsoleUI` 部件和相关变量』

第 565 页的『`openUI`』

相关任务

第 164 页的『使用 `consoleUI` 创建界面』

EGL consoleUI 中的 Menu 字段

以下列表定义类型为 `Menu` 的变量中的各个字段。必须指定字段 **`labelText`** 或 **`labelTextKey`**。

labelText

显示在 `menuItem` 列表左边的标签。

类型: 字符串文字

示例: *labelText = "Options: "*

缺省值: *none*。

在运行时是否可更新? 否

labelKey

指定用于搜索资源束的键, 该资源束包含菜单标签。如果同时指定 **`labelText`** 和 **`labelKey`**, 则使用 **`labelText`**。

类型: *String*

示例: `labelKey = "myKey"`

缺省值: `Empty string`

在运行时是否可更新? 否

资源束是由系统变量 `ConsoleLib.messageResource` 标识的, 如 `messageResource` 中所述。

menuItem

菜单项数组, 每个菜单项是在程序中声明的或者是使用关键字 `new` 以动态方式创建的。有关第二种选择的详细信息, 请参阅 `consoleUI` 中 `new` 的用法。

类型: `MenuItem[]`

示例: `menuItems = [myItem, new MenuItem {name = "Remove", labelText = "Delete all"}]`。

缺省值: `none`。

在运行时是否可更新? 否

可使用以下语法在程序中添加 `menuItem`:

```
myMenu.MenuItems.addElement(myMenuItem)
```

`myMenu`

类型为 `Menu` 的变量的名称。

`myMenuItem`

类型为 `MenuItem` 的变量的名称。

如果对没有 `menuItem` 的菜单发出 `openUI` 语句, 则程序终止。

相关概念

第 163 页的『控制台用户界面』

相关参考

第 68 页的『数组』

第 165 页的『ConsoleUI 部件和相关变量』

第 565 页的『openUI』

『EGL consoleUI 中的 MenuItem 字段』

第 168 页的『ConsoleUI 中 new 的用法』

相关任务

第 164 页的『使用 consoleUI 创建界面』

EGL consoleUI 中的 MenuItem 字段

以下列表定义类型为 `MenuItem` 的变量中的各个 `consoleField`。所有 `consoleField` 都不是必需的; 可通过设置下列三个字段中的任何一个来确定用户的选择: `accelerators`、`labelText` 或 `labelKey`。

accelerators

指示相当于用户选择的 `menuItem` 的击键。每个击键都会执行对应于该 `menuItem` 选择的 `openUI` 语句的 `OnEvent` 子句。

类型: `String[]`

示例: `accelerators = ["F1", "ALT_F1"]`

缺省值: *none*

在运行时是否可更新? 否

comment

指定注释, 这是选择 `menuItem` 时在特定于 `menuItem` 的注释行中显示的文本。

类型: *String*

示例: *"Delete the record"*

缺省值: *Empty string*

在运行时是否可更新? 是

注释行就是菜单行下面的一行。

commentKey

指定用于搜索包括注释的资源束的键, 注释是选择 `menuItem` 时在特定于 `menuItem` 的注释行 (如果有的话) 中显示的文本。如果同时指定 **comment** 和 **commentKey**, 则使用 **comment**。

类型: *String*

示例: *commentKey = "myKey"*

缺省值: *Empty string*

在运行时是否可更新? 是

资源束是由系统变量 **ConsoleLib.messageResource** 标识的, 如 *messageResource* 中所述。

help

指定出现以下情况时要显示的文本:

- 选择了该 `menuItem`; 并且
- 用户按了 **ConsoleLib.key_help** 中标识的键。

类型: *String*

示例: *help = "Deletion is permanent"*

缺省值: *Empty string*

在运行时是否可更新? 是

helpKey

指定用于搜索资源束的访问键, 该资源束包含在出现以下情况时将显示的文本:

- 选择了该 `menuItem`; 并且
- 用户按了 **ConsoleLib.key_help** 中标识的键。

如果同时指定 **help** 和 **helpKey**, 则使用 **help**。

类型: *String*

示例: *helpKey = "myKey"*

缺省值: *Empty string*

在运行时是否可更新? 是

资源束是由系统变量 **ConsoleLib.messageResource** 标识的, 如 *messageResource* 中所述。

labelText

表示 `menuItem` 的标签。

类型: *String literal*

示例: `labelText = "Delete"`.

缺省值: *none*。

在运行时是否可更新? 否

labelKey

指定用于搜索资源束的键, 该资源束包含 `menuItem` 标签。如果同时指定 **labelText** 和 **labelKey**, 则使用 **labelText**。

类型: *String*

示例: `labelKey = "myKey"`

缺省值: *Empty string*

在运行时是否可更新? 否

资源束是由系统变量 **ConsoleLib.messageResource** 标识的, 如 `messageResource` 中所述。

name

`MenuItem` 名称, 在运行时解析名称的编程环境中使用。另外, 该名称将在对应于 `menuItem` 选择的 **openUI** 语句中使用。

建议名称字段的值与变量名称相同。

类型: *String*

示例: `name = "myItem"`

缺省值: *none*

在运行时是否可更新? 否

相关概念

第 163 页的『控制台用户界面』

相关参考

第 165 页的『ConsoleUI 部件和相关变量』

第 414 页的『EGL consoleUI 中的 Menu 字段』

第 565 页的『openUI』

相关任务

第 164 页的『使用 consoleUI 创建界面』

EGL consoleUI 中的 PresentationAttributes 字段

以下列表定义可在类型为 `PresentationAttributes` 的任意系统变量中设置或检索的字段:

color

指定颜色:

类型: *ColorKind*

示例: `color = red`

缺省值: *white*

在运行时是否可更新? 是, 但仅对在更新颜色字段后显示的输出才会有更新的视觉效果

值如下所示:

defaultColor 或 **white** (缺省值)

白色

black

黑色

blue

蓝色

cyan

青色

green

绿色

magenta

品红色

red

红色

yellow

黄色

highlight

指定显示输出时要使用的特殊效果 (如果有的话)。

类型: *HighlightKind[]*

示例: *highlight = [reverse, underline]*

缺省值: *[noHighLight]*

在运行时是否可更新? 是, 但仅对在更新突出显示字段后显示的输出才会有更新的视觉效果

值如下所示:

noHighlight (缺省值)

不会有特殊效果。使用此值将覆盖任何其它值。

blink

现在此值不生效。

reverse

反转文本和背景色, 这样的话 (举例来说), 如果显示器是黑底白字的, 则背景变为白色的, 而文本变为黑色的。

underline

在受影响区域下面加下划线。下划线的颜色就是文本的颜色, 即使同时指定了值 **reverse** 也是如此。

intensity

指定显示字体的强度。

类型: *IntensityKind[]*

示例: *intensity = [bold]*

缺省值: *[normalIntensity]*

在运行时是否可更新？ 是，但是仅对于更新强度字段后显示的输出才会有更新的视觉效果

值如下所示：

normalIntensity (缺省值)

不会有特殊效果。使用此值将覆盖任何其它值。

bold

使文本以粗体字体显示。

dim

现在此值不生效。将来，所有输入字段被禁用时，可以适当地让文本以较低强度出现。

invisible

除去任何有关“文本在表单上”的指示。

相关概念

第 163 页的『控制台用户界面』

相关参考

第 701 页的『currentDisplayAttrs』

第 701 页的『currentRowAttrs』

第 701 页的『defaultDisplayAttributes』

第 702 页的『defaultInputAttributes』

第 165 页的『ConsoleUI 部件和相关变量』

第 565 页的『openUI』

相关任务

第 164 页的『使用 consoleUI 创建界面』

EGL consoleUI 中的 Prompt 字段

以下列表定义类型为 Prompt 的变量中的各个字段。所有字段都不是必需的。

isChar

指示在显示提示后用户的第一次击键是否结束该操作。

类型: *Boolean*

示例: *isChar = yes*

缺省值: *no*

在运行时是否可更新？ 是

值如下所示：

no (缺省值)

当用户按 **Enter** 或按与显示该提示的 **openUI** 语句的 **OnEvent** 子句相关联的键时，该操作结束。与该提示绑定的变量接收输入字符。

yes

用户的第一次击键结束该操作。如果字符是可打印字符，则与该提示绑定的变量将接收该字符。

在任一情况下，可通过设置类型为 **ON_KEY** 的 **OnEvent** 子句来响应特定击键。

message

指定提示用户的文本。

类型: *String*

示例: `message = "Type here: "`

缺省值: *Empty string*

在运行时是否可更新? 是, 在代码发出 `openUI` 语句之前

messageKey

指定用来搜索包括提示文本的资源束的键。如果同时指定 `message` 和 `messageKey`, 则使用 `message`。

类型: *String*

示例: `messageKey = "promptText"`

缺省值: *Empty string*

在运行时是否可更新? 是

资源束是由系统变量 `ConsoleLib.messageResource` 标识的, 如 `messageResource` 中所述。

responseAttr

指定在显示用户输入时使用的表示属性。

类型: *PresentationAttributes literal*

示例: `responseAttr {color = green, highlight = [underline], intensity = [bold]}`

缺省值: *no*

在运行时是否可更新? 是

仅当字段 `isChar` 设置为 *no* 时, 此字段才起作用。

有关 `responseAttr` 值的详细信息, 请参阅 `EGL consoleUI` 中的 `PresentationAttributes` 字段。

相关概念

第 163 页的『控制台用户界面』

相关参考

第 165 页的『ConsoleUI 部件和相关变量』

第 493 页的『Java 运行时属性 (详细信息)』

第 714 页的『messageResource』

第 565 页的『openUI』

第 417 页的『EGL consoleUI 中的 PresentationAttributes 字段』

相关任务

第 164 页的『使用 consoleUI 创建界面』

EGL consoleUI 中的 Window 字段

以下列表定义类型为 `Window` 的变量中的各个字段。所有字段都不是必需的, 但在实际操作中需要 `size`。

color

指定在窗口中显示下列种类的输出时使用的颜色:

- `consoleForm` 中的标签

- 提示中的输入字段
- 窗口边框
- **ConsoleLib.displayAtPosition** 之类的系统函数的输出

类型: *ColorKind*

示例: *color = red*

缺省值: *white*

在运行时是否可更新? 是, 但仅当在更新字段后打开窗口时才会有更新的视觉效果

值如下所示:

defaultColor 或 **white** (缺省值)

白色

black

黑色

blue

蓝色

cyan

青色

green

绿色

magenta

品红色

red

红色

yellow

黄色

commentLine

在 Window 字段 **hasCommentLine** 设置为 *yes* 时用于设置显示注释 (如果有的话) 的行号。行号从屏幕窗口的内容区域的顶部开始计算 (在这种情况下第一行的行号为 1) 或者 (如果该值为负数) 从该区域的底部开始计算 (在这种情况下, 最后一行为 -1, 倒数第二行为 -2, 以此类推)。

类型: *INT*

示例: *commentLine = 10*

缺省值: 窗口的最后一行 (尽管只要屏幕窗口是打开的, 注释就在该窗口的倒数第二行上)

在运行时是否可更新? 是, 但仅当在更新字段后打开窗口时才会有更新的视觉效果

该值是否有效只能在运行时确定。

formLine

设置显示表单的行号。行号从屏幕窗口的内容区域的顶部开始计算 (在这种情况下第一行的行号为 1) 或者 (如果该值为负数) 从该区域的底部开始计算 (在这种情况下, 最后一行为 -1, 倒数第二行为 -2, 以此类推)。

类型: *INT*

示例: *formLine = 8*

缺省值: *3*

在运行时是否可更新? 是, 但仅当在更新字段后显示窗口时才会有更新的视觉效果

该值是否有效只能在运行时确定。

hasBorder

指示窗口周围是否有边框。如果该值为 *yes*, 则边框的颜色是在 *Window* 字段 *color* 中指定的。

类型: *Boolbean*

示例: *hasBorder = yes*

缺省值: *no*

在运行时是否可更新? 是, 但仅当在更新字段后打开窗口时才会有更新的视觉效果

hasCommentLine

指示窗口是否保留一行以供注释使用, 注释是光标进入 *consoleField* 时显示的文本条目。如果该值为 *yes*, 则行号是在 *Window* 字段 *commentLine* 中指定的。

类型: *Boolbean*

示例: *hasCommentLine = yes*

缺省值: *no*

在运行时是否可更新? 是, 但仅当在更新字段后打开窗口时才会有更新的视觉效果

highlight

指定在窗口中显示下列种类的输出时使用的特殊效果 (如果有的话):

- *consoleForm* 中的标签
- 提示中的输入字段
- 窗口边框
- **ConsoleLib.displayAtPosition** 之类的系统函数的输出

类型: *HighlightKind[]*

示例: *highlight = [reverse, underline]*

缺省值: *[noHighLight]*

在运行时是否可更新? 是, 但仅当在更新字段后显示窗口时才会有更新的视觉效果

值如下所示:

noHighlight (缺省值)

不会有特殊效果。使用此值将覆盖任何其它值。

blink

现在此值不生效。

reverse

反转文本和背景色, 这样的话 (举例来说), 如果显示器是黑底白字的, 则背景变为白色的, 而文本变为黑色的。

underline

在受影响区域下面加下划线。下划线的颜色就是文本的颜色，即使因为同时指定了值 **Reverse** 而导致文本颜色反转也是如此。

intensity

指定在窗口中显示下列种类的输出时使用的显示字体的强度：

- consoleForm 中的标签
- 提示中的输入字段
- 窗口边框
- **ConsoleLib.displayAtPosition** 之类的系统函数的输出

类型: *IntensityKind[]*

示例: *intensity = [bold]*

缺省值: *[normalIntensity]*

在运行时是否可更新? 是，但仅当在更新字段后打开窗口时才会有更新的视觉效果

值如下所示：

normalIntensity (缺省值)

不会有特殊效果。使用此值将覆盖任何其它值。

bold

使文本以粗体字体显示。

dim

现在此值不生效。将来，所有输入字段被禁用时，可以适当地让文本以较低强度出现。

invisible

除去任何有关“字段位于表单中”的指示。

menuLine

设置在窗口中显示菜单（如果有的话）的行号。行号从屏幕窗口的内容区域的顶部开始计算（在这种情况下第一行的行号为 1）或者（如果该值为负数）从该区域的底部开始计算（在这种情况下，最后一行为 -1，倒数第二行为 -2，以此类推）。

类型: *INT*

示例: *menuLine = 2*

缺省值: *1*

在运行时是否可更新? 是，但仅当在更新字段后打开窗口时才会有更新的视觉效果

该值是否有效只能在运行时确定。

messageLine

设置在窗口中显示消息（如果有的话）的行号。行号从屏幕窗口的内容区域的顶部开始计算（在这种情况下第一行的行号为 1）或者（如果该值为负数）从该区域的底部开始计算（在这种情况下，最后一行为 -1，倒数第二行为 -2，以此类推）。

类型: *INT*

示例: *messageLine = 3*

缺省值: *2*

在运行时是否可更新？ 是， 但仅当在更新字段后打开窗口时才会有更新的视觉效果

该值是否有效只能在运行时确定。

name

窗口名称， 在运行时解析名称的编程环境中使用。 建议名称字段的值与变量名称相同。

类型: *String*

示例: *name = "myWindow"*

缺省值: *none*

在运行时是否可更新？ 否

position

屏幕窗口的内容区域中的窗口左上角位置。 该字段包含由两个整数组成的数组： 行号和跟在行号后面的列号。 行号从屏幕窗口的内容区域的顶部开始计算（在这种情况下第一行的行号为 1）或者（如果该值为负数）从该区域的底部开始计算（在这种情况下， 最后一行为 -1， 倒数第二行为 -2， 以此类推）。 列号是从控制台窗口的内容区域的左边开始计算， 第一列为 1。

类型: *INT[2]*

示例: *position = [2, 3]*

缺省值: *[1,1]*

在运行时是否可更新？ 否

promptLine

设置在窗口中显示提示（如果有的话）的行号。 行号从控制台窗口的内容区域的顶部开始计算或者（如果该值为负数）从该区域的底部开始计算。

类型: *INT*

示例: *promptLine = 4*

缺省值: *1*

在运行时是否可更新？ 是， 但仅当在更新字段后打开窗口时才会有更新的视觉效果

该值是否有效只能在运行时确定。

size

由两个正整数组成的数组， 表示窗口维： 行数和跟在行数后面的列数。

类型: *INT[2]*

示例: *size = [24, 80]*

缺省值: *none*

在运行时是否可更新？ 否

实际操作中必需有一个值。 如果显示的窗口缺少表示 **size** 的值， 则运行时显示的窗口就会太小， 无法显示内容。

如果维超出屏幕窗口的内容区域中的可用大小， 则在运行时会发生错误。

相关概念

第 163 页的『控制台用户界面』

相关参考

第 165 页的『ConsoleUI 部件和相关变量』

第 565 页的『openUI』

相关任务

第 164 页的『使用 consoleUI 创建界面』

containerContextDependent

函数部件属性 **containerContextDependent** 允许您扩展用于解析包括该属性的函数部件中的函数引用的名称空间。有效值为 *no*（缺省值）和 *yes*。

在开发新代码时，建议您不要使用此功能。此属性主要用于从 VisualAge Generator 迁移程序。但是，如果将此属性设置为 *yes*，则含义如下：

- 如果一般的名称搜索步骤在编辑时无法解析引用，则 EGL 编辑器不将未解析的引用标志为错误。
- 如果一般的名称搜索步骤在生成时无法解析引用，则通过查看包含或函数部件的程序、库或 PageHandler 的名称空间来继续搜索。
- 如果在 EGL 源文件的顶部（而不是完全在容器内部，即程序、PageHandler 或库）声明了一个函数，仅当出现下列情况时，该函数才能调用库函数：
 - 容器包括引用库的 `use` 语句
 - 在调用函数中，属性 **containerContextDependent** 设置为 *yes*

相关概念

第 20 页的『对部件的引用』

相关参考

第 481 页的『EGL 源格式的函数部件』

第 875 页的『使用声明』

数据库权限和表名

授权标识是一个字符串，当在数据库管理器与程序之间建立连接时，该字符串被传递给数据库管理器，而无论程序是准备另一个程序还是允许最终用户访问 SQL 表。该字符串是检查由准备者或最终用户所拥有的数据库访问权限所需的用户标识。

授权标识的源取决于进行数据库访问的系统。

对于 EGL 生成的 Java 程序，情况如下所示：

- 授权标识是在数据库管理器与程序之间建立连接时由数据库管理器获取的标识：
 - 对于缺省数据库来说，授权标识是对 Java 运行时属性 **vgj.jdbc.default.userid** 指定的值
 - 当调用系统函数 **sysLib.connect** 或 **VGLib.connectionService** 时，授权标识是对 `userID` 参数指定的值

当您指定表名时可能会使用授权标识。在该情况下，可以根据以下语法来指定表名限定符：

```
tableOwner.myTable
```

tableOwner

这是数据库管理器知道的并且标识表所需要的限定符。创建表时的限定符就是创建表的人员的授权标识。

myTable

表名。

有关授权标识的更多信息，请参阅数据库管理器文档。

相关概念

第 219 页的『动态 SQL』

第 315 页的『Java 运行时属性』

第 209 页的『SQL 支持』

相关参考

第 493 页的『Java 运行时属性（详细信息）』

第 683 页的『EGL 源格式的 SQL 记录部件』

第 820 页的『connect()』

第 839 页的『connectionService()』

数据转换

由于在不同运行时环境中解释数据的方式存在差别，因此，程序可能需要对从一种环境传递到另一种环境的数据进行转换。数据转换是在 Java 运行时发生的。

在下列运行时情况下，代码也将使用转换表：

- EGL 生成的 Java 代码调用 CICS for z/OS 上的程序。

在这种情况下，可以在引用被调用程序的 `callLink` 元素中指定转换表。另外，也可以（在该 `callLink` 元素中）指示系统变量 `sysVar.callConversionTable` 在运行时标识转换表。

- EGL 生成的程序（位于支持 EBCDIC 字符集的平台）以异步方式转移到位于支持 ASCII 字符集的平台上的程序，当转移程序调用系统函数 `sysLib.startTransaction` 时，就会发生这种情况。

在这种情况下，可以在引用得到控制权的程序的 `asynchLink` 元素中指定转换表。另外，也可以（在该 `asynchLink` 元素中）指示系统变量 `sysVar.callConversionTable` 在运行时标识转换表。

- EGL 生成的 Java 程序显示一个文本表单或打印表单，该文本表单或打印表单包含一系列阿拉伯语或希伯来语字符；或者显示一个文本表单，该文本表单从用户那里接受一系列此类字符。

在这些情况下，在系统变量 `sysVar.formConversionTable` 中指定双向转换表。

例如，如果代码将值放在两个重新定义的记录的其中一条记录中，每个记录都与被传递给另一程序的记录引用同一个内存区，则将使用运行时转换。假定传递的数据的特征根据被赋值的重新定义的记录的不同而有所不同。在这种情况下，在生成时不能了解数据转换的需求。

以下各节提供了下列详细信息:

-
- 『当调用程序是 Java 代码时的数据转换』
- 第 428 页的『转换算法』

当调用程序是 Java 代码时的数据转换

下列规则适用于 Java 代码:

- 当生成的 Java 程序或包装器调用生成的 Java 程序时, 就会按照在运行时调用的一组 EGL 类来在调用者中进行转换。在大多数情况下, 根本就不需要请求转换, 即使调用程序正在访问的远程平台使用的代码页与调用程序使用的代码页不同。然而在下列情况下, 必须指定转换表:
 - 调用程序是 Java 代码并且位于支持一种代码页的机器上
 - 被调用程序为非 Java 程序, 并且位于支持另一种代码页的机器上

在这种情况下, 表名是一个指示在运行时所需要的转换类型的符号。

- 当生成的 Java 程序访问远程 MQSeries 消息队列时, 会根据运行时调用的一组 EGL 类来在调用程序中进行转换。如果调用程序正在访问的远程平台使用的代码页与调用程序使用的代码页不同, 则在引用 MQSeries 消息队列的关联元素中指定转换表。

下表列示生成的 Java 代码在运行时可访问的转换表。每个名称都具有格式 CSOJx:

- x 表示被调用平台上的代码页号。每个号码都是在 *Character Data Representation Architecture Reference and Registry* (SC09-2190) 中指定的。该注册表标识了转换表支持的编码字符集。

语言	被调用程序的平台		
	UNIX	Windows 2000/NT/XP	z/OS UNIX System Services 或 iSeries Java
阿拉伯语	CSOJ1046	CSOJ1256	CSOJ420
简体中文	CSOJ1381	CSOJ1386	CSOJ1388
繁体中文	CSOJ950	CSOJ950	CSOJ1371
Cyrillic	CSOJ866	CSOJ1251	CSOJ1025
丹麦语	CSOJ850	CSOJ850	CSOJ277
东欧语系	CSOJ852	CSOJ1250	CSOJ870
英国英语	CSOJ850	CSOJ1252	CSOJ285
美国英语	CSOJ850	CSOJ1252	CSOJ037
法语	CSOJ850	CSOJ1252	CSOJ297
德语	CSOJ850	CSOJ1252	CSOJ273
希腊语	CSOJ813	CSOJ1253	CSOJ875
希伯来语	CSOJ856	CSOJ1255	CSOJ424
日语	CSOJ943	CSOJ943	CSOJ1390 (片假名 SBCS) 和 CSOJ1399 (拉丁语 SBCS)
韩国语	CSOJ949	CSOJ949	CSOJ1364

语言	被调用程序的平台		
	UNIX	Windows 2000/NT/XP	z/OS UNIX System Services 或 iSeries Java
葡萄牙语	CSOJ850	CSOJ1252	CSOJ037
西班牙语	CSOJ850	CSOJ1252	CSOJ284
瑞典语	CSOJ850	CSOJ1252	CSOJ278
瑞士德语	CSOJ850	CSOJ1252	CSOJ500
土耳其语	CSOJ920	CSOJ1254	CSOJ1026

如果从 Java 中调用程序时未在链接选项部件中指定转换表值，则缺省转换表是用于英语（美国）的转换表。

转换算法

记录和结构的数据转换是根据缺少子结构的结构项的声明来进行的。

类型为 CHAR、DBCHAR 或 MBCHAR 的数据是根据 Java 转换表（用于在 EGL 生成的调用程序中进行的转换）来转换的。

不会对填充符数据项（没有名称的数据项）或类型为 DECIMAL、PACF、HEX 或 UNICODE 的数据项执行任何转换。

在对 MBCHAR 数据进行的从 EBCDIC 至 ASCII 的转换中，转换例程将删除 Shift-out/Shift-in (SO/SI) 字符，并在数据项末尾插入相同数目的空格。在从 ASCII 至 EBCDIC 的转换中，转换例程在双字节字符串两边插入 SO/SI 字符，并在可填写到字段中的最后一个有效字符处截断值。如果 MBCHAR 字段在变长记录中并且当前记录结尾位于 MBCHAR 字段中，则会调整记录长度以反映插入或删除 SO/SI 字符。记录长度指示当前记录在何处结束。

如果调用程序或被调用程序的平台使用 Intel 二进制格式而其它平台不使用该格式，则对于类型为 BIN 的数据项，转换例程会使项的字节顺序相反。

对于类型为 NUM 或 NUMC 的数据项，转换例程会使用 CHAR 算法来转换除了最后一个字节之外的所有字节。根据下表中显示的十六进制值来转换符号半字节（字段中最后一个字节的前半个字节）。

类型为 NUM 的 EBCDIC	类型为 NUMC 的 EBCDIC	ASCII
F (正号)	C	3
D (负号)	D	7

相关参考

第 340 页的『关联元素』

第 429 页的『双向语言文本』

第 370 页的『callLink 元素』

第 822 页的『convert()』

第 852 页的『callConversionTable』

双向语言文本

双向 (bidi) 语言 (如阿拉伯语和希伯来语) 是按照从右到左的顺序向用户显示文本的语言, 但是文本中的数字和拉丁字母字符串是从左到右显示的。另外, 字符在程序变量中的显示顺序是可以改变的。文本通常按逻辑顺序进行存储, 也就是在输入字段中输入字符的顺序。

在排序以及其它相关联的显示特征方面的差别要求程序具有将双向文本字符串从一种格式转换为另一种格式的能力。bidi 转换属性是在 bidi 转换表 (.bct) 文件中指定的, 该文件是独立于程序创建的。程序引用转换表的名称以指示应该如何执行属性转换。

在所有情况下, 都将 bidi 转换表引用指定为 1 到 8 个字符的文件名 (不带扩展名 .bct)。例如, 如果已经创建了一个名为 mybct.bct 的 bidi 转换表, 则在程序中可以通过在程序开头添加以下语句来设置 formConversionTable 值:

```
sysVar.formConversionTable = "mybct.bct" ;
```

您的任务如下所示:

- 创建用于指定应该进行的变换的 bidi 转换表。注意, 转换的文本或打印表单中显示的数据需要不同的表。
- 当生成使用带有 bidi 语言文本的文本表单或打印表单的程序时, 请对程序添加一个语句, 该语句在显示表单之前将转换表名赋值给系统函数 sysVar.formConversionTable。

使用 bidi 转换表向导插件来构建 bidi 转换表文件, 该插件位于 BidiConversionTable.zip 文件中:

1. 从以下 Web 站点下载该文件:
2. 将该文件解压缩到工作台目录中
3. 要开始运行向导, 单击 **文件 > 新建 > 其它 > BidiConversionTable**。

与 EGL 程序配合使用的表的名称必须具有 8 个 (或少于 8 个) 字符并且必须具有 .bct 扩展名。

4. 运行向导时, 可以按 F1 键获得有关选择用于创建表的正确选项的帮助。

相关参考

第 426 页的『数据转换』

第 852 页的『callConversionTable』

数据初始化

如果 EGL 生成的程序自动初始化记录（在某些情况下会发生这种情况，在后文中有所描述），则每个最低级别结构项都被设置为适合于基本类型的值。表单初始化也类似，但表单声明可以指定用于覆盖缺省值的值。

在下列情况下也会发生初始化：

- 变量（确切地说，项、记录或静态数组）的 **initialized** 属性被设置为 *yes*。
- 逻辑包含 `set` 的特定变体

下表提供了有关初始化值的详细信息。

基本类型	初始化值
ANY	变量属于未定义类型
BIN 和整数类型（BIGINT、INT 和 SMALLINT）、HEX、FLOAT 和 SMALLFLOAT	二进制零
CHAR 和 MBCHAR	单字节空格
DATE、TIME 和 TIMESTAMP	机器时钟的当前值（对于 TIMESTAMP，表示掩码所需的字节数）
DBCHAR	双字节空格
DECIMAL、MONEY、NUM、NUMC 和 PACF	数字 0
INTERVAL	以加号开头的数字零（表示掩码所需的字节数）
UNICODE	Unicode 空格（每个都是十六进制的 0020）

在结构中，只考虑最低级别的结构项。例如，如果 HEX 类型的结构项是 CHAR 类型的结构项的下级，则使用二进制零来初始化内存区。

作为程序或函数参数接收的记录或项不会被自动初始化。

EGL 生成的程序初始化记录，无论是局部记录还是全局记录。

相关概念

- 第 130 页的『函数部件』
- 第 121 页的『DataItem 部件』
- 第 128 页的『程序部件』
- 第 122 页的『记录部件』
- 第 23 页的『固定结构』

相关参考

- 第 80 页的『EGL 语句』

- 第 579 页的『set』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 448 页的『EGL 源格式』

第 481 页的『EGL 源格式的函数部件』

第 488 页的『EGL 源格式的带索引记录部件』

第 603 页的『EGL 源格式的 MQ 记录部件』

第 612 页的『命名约定』

第 31 页的『基本类型』

第 664 页的『EGL 源格式的程序部件』

第 676 页的『EGL 源格式的相关记录部件』

第 679 页的『EGL 源格式的串行记录部件』

第 683 页的『EGL 源格式的 SQL 记录部件』

EGL 源格式的 DataTable 部件

可以在 EGL 文件中声明 `dataTable` 部件，*EGL 项目*、*包和文件*对该部件作了描述。此部件是可生成部件，这意味着它必须位于文件的顶层，并且必须与文件同名。

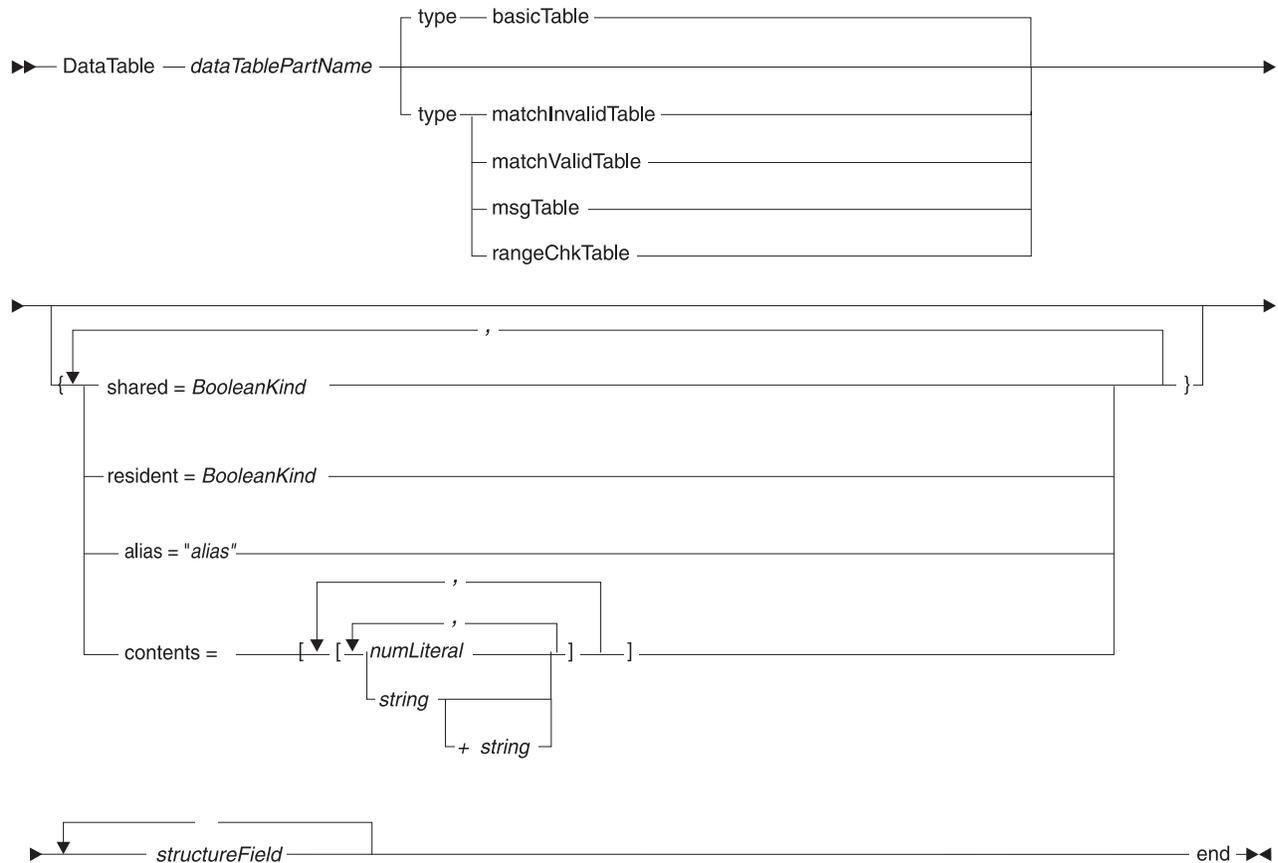
`dataTable` 通过程序的使用声明或（作为程序唯一的消息表时）通过程序的 **`msgTablePrefix`** 属性与程序相关。`dataTable` 通过 `pageHandler` 的使用声明与 `pageHandler` 相关。

以下是 `dataTable` 部件的一个示例：

```
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
```

`dataTable` 部件的语法图如下所示：



DataTable *dataTablePartName* ... end

将部件标识为 `dataTable` 并指定部件名。有关命名规则，请参阅 [命名约定](#)。

basicTable (缺省值)

包含程序逻辑所使用的信息；例如，国家或地区以及相关代码的列表。

matchInvalidTable

在文本字段的 `validatorDataTable` 属性中指定，它指示用户的输入必须与 `dataTable` 的第一列中的任何值不相同。作为对验证失败的响应，EGL 运行时执行以下操作：

- 访问 `validatorDataTable` 属性中引用的表
- 检索由特定于文本字段的 `validatorDataTableMsgKey` 属性标识的消息
- 显示由特定于表单的 `msgField` 属性标识的文本字段中的消息

matchValidTable

在文本字段的 `validatorDataTable` 属性中指定，它指示用户的输入必须与 `dataTable` 中的第一列中的值匹配。作为对验证失败的响应，EGL 运行时执行以下操作：

- 访问 `validatorDataTable` 属性中引用的表
- 检索由特定于文本字段的 `validatorDataTableMsgKey` 属性标识的消息
- 显示由特定于表单的 `msgField` 属性标识的字段中的消息

msgTable

包含运行时消息。在下列情况下将显示一条消息：

- 表是程序的消息表。如果程序属性 **msgTablePrefix** 引用表前缀（这是 dataTable 的名称的第一到第四个字符），则表与程序发生关联。名称的其余部分是下表中的其中一个本地语言代码。

语言	本地语言代码
巴西葡萄牙语	PTB
简体中文	CHS
繁体中文	CHT
大写英语	ENP
美国英语	ENU
法语	FRA
德语	DEU
意大利语	ITA
日语，片假名（单字节字符集）	JPN
韩国语	KOR
西班牙语	ESP
瑞士德语	DES

- 程序通过两种机制的其中一种来检索和显示消息，如 *ConverseLib.displayMsgNum* 和 *ConverseLib.validationFailed* 所述。

rangeChkTable

在文本字段的 **validatorDataTable** 属性中指定，它指示用户的输入必须与至少介于一个数据表行的第一列和第二列的值之间的值匹配。（范围是包括边界的；当用户的输入与任何行的第一列或第二列的值相匹配时，输入有效。）作为对验证失败的响应，EGL 运行时执行以下操作：

- 访问 **validatorDataTable** 属性中引用的表
- 检索由特定于文本字段的 **validatorDataTableMsgKey** 属性标识的消息
- 显示由特定于表单的 **msgField** 属性标识的字段中的消息

"alias"

一个字符串，它包含在生成的输出的名称中。如果未指定别名，则会使用 dataTable 名。

shared

指示 dataTable 的同一个实例是否被同一个运行单元中的多个程序使用。有效值是 *yes* 和 *no*（缺省值）。如果 **shared** 的值是 *no*，则运行单元中的每个程序都拥有 dataTable 的唯一副本。

此属性指示 dataTable 的同一个实例是否被同一运行单元中的每个程序使用。如果 **shared** 的值是 *no*，则运行单元中的每个程序都拥有 dataTable 的唯一副本。

在运行时进行的更改对每个能够访问 dataTable 的程序都是可视的，而且这些更改将保留到 dataTable 被卸装为止。在大多数情况下，**resident** 属性（在后面描述）的值确定何时卸装 dataTable；有关详细信息，请参阅该属性的描述。

resident

指示在每个访问 dataTable 的程序都已结束之后，是否仍将 dataTable 存放在内存中。

有效值是 *yes* 和 *no*。缺省值为 *no*。

如果将 **resident** 属性设置为 *yes*，则将共享 `dataTable`，而不考虑 **shared** 的值。

使 `dataTable` 驻留下来有以下好处：

- `dataTable` 保留了先前运行的程序对其写入的任何值
- 可以立即访问该表，而不需要进行附加的装入处理

在运行单元结束之前，驻留的 `dataTable` 一直保持已装入状态。但是，当使用非驻留 `dataTable` 的程序结束时，非驻留 `dataTable` 将被卸装。

注：`dataTable` 是在第一次被程序访问时而不是在 EGL 运行时处理使用声明时装入内存的（如果必要的话）。

contents

`dataTable` 单元格的值，每个值都为下列其中一种类型：

- 数字文字
- 字符串文字或字符串文字的并置

给定行中的内容的类型必须与顶层结构字段兼容，每个结构字段都表示一个列定义。

structureField

结构字段，如 *EGL* 源格式的结构字段中所述。

相关概念

第 134 页的『`DataTable`』

第 13 页的『EGL 项目、包和文件』

第 678 页的『运行单元』

相关参考

第 612 页的『命名约定』

第 686 页的『EGL 源格式中的结构字段』

第 721 页的『`displayMsgNum()`』

第 722 页的『`validationFailed()`』

第 875 页的『使用声明』

EGL 构建路径和 `eglp`

每个 EGL 项目和 EGL Web 项目都与一个 EGL 构建路径相关联，因此，项目可以引用其它项目中的部件。有关何时使用 EGL 构建路径以及有关构建路径条目顺序为何重要的详细信息，请参阅对部件的引用。

当您指定 EGL 构建路径时，可以选择导出构建路径中列示的一个或多个项目。于是，当某个项目引用正在声明的项目时，已导出的每个项目都可供引用项目使用，如以下示例所示：

- 项目 A 的 EGL 构建路径由下列项目按顺序组成：

A, B, C, D

已将项目 B 和 D 导出。

- 项目 L 的 EGL 构建路径由下列项目按顺序组成：

L, J, A, Z

- 项目 L 的有效构建路径也包含已从项目 A 中导出的项目。在这种情况下，项目 L 的 EGL 构建路径是按照以下方式发挥作用的：

L, J, A, B, D, Z

导出的项目被放在导出它们的项目之后，并且它们的顺序就是这些项目在导出项目的构建路径中的列示顺序。

项目的构建路径始终包含该项目本身，该项目通常是构建路径顺序中的第一个项目，这也是建议的做法。如果在项目中有多个 EGL 源文件夹，则必须将所有那些文件夹列示在该项目的 EGL 构建路径中，并且任何引用该项目的项目都将使用那些文件夹的顺序。

强烈建议您避免在不同的项目中或在同一个项目的不同文件夹中放置名称完全相同的包。

如果在 EGL SDK 中生成，则情况如下所示：

- 没有项目信息可用。
- 命令行自变量 *eglp* 替换了 EGL 构建路径的功能。*eglp* 是操作系统目录列表，EGL SDK 在尝试解析部件引用时将搜索这些目录。
- 何时使用 *eglp* 的规则与何时使用 EGL 构建路径的规则相同；但是，与您可以导出项目不同，不能导出目录。

当使用 *EGL SDK* 时，强烈建议您避免在不同的目录中放置名称完全相同的包。

相关概念

第 302 页的『从 EGL 软件开发包 (SDK) 生成』

第 20 页的『对部件的引用』

相关任务

第 302 页的『从 EGL 软件开发包 (SDK) 生成』

相关参考

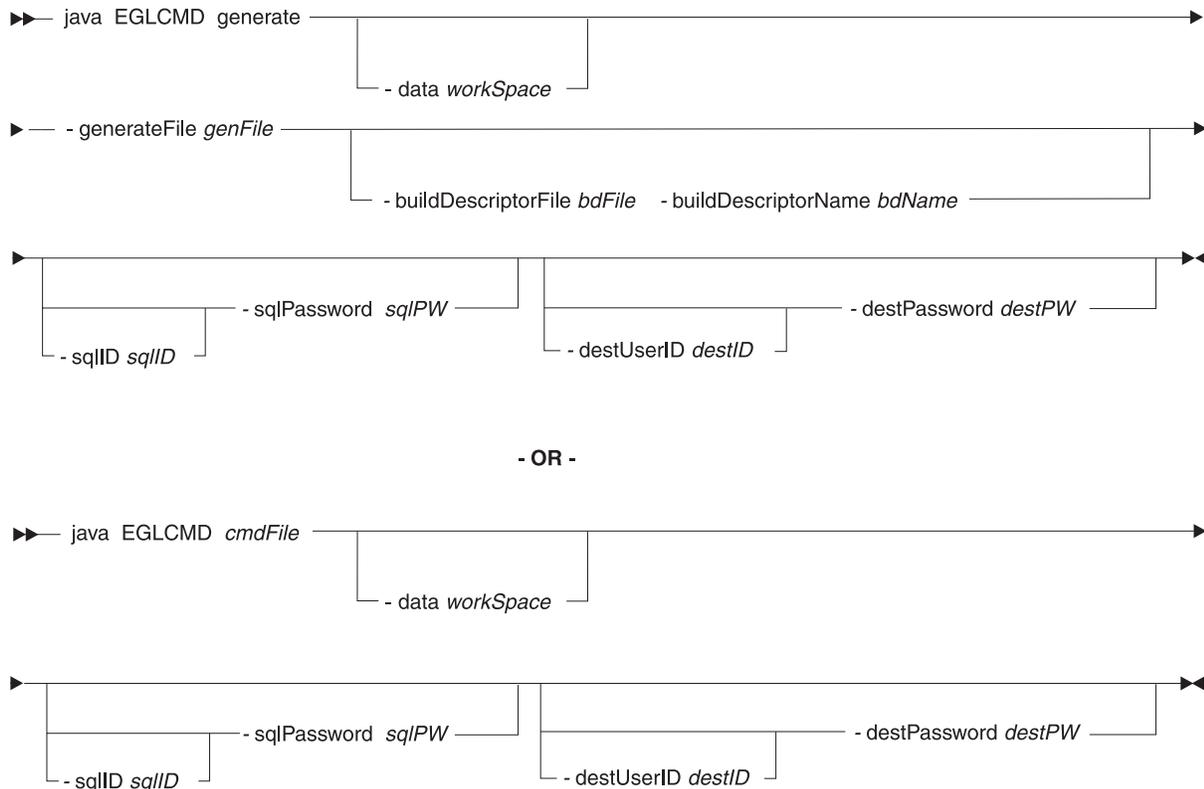
第 446 页的『EGLSDK』

EGLCMD

命令 *EGLCMD* 使您能够访问工作台批处理接口，如从工作台批处理接口生成中所述。

语法

EGLCMD 的调用语法如下所示：



generate

指示该命令本身引用用来生成输出的 EGL 文件和构建描述符部件。在此例中，命令 EGLCMD 不引用命令文件。

-data *workSpace*

指定工作空间目录的绝对路径或相对路径。相对路径相对于您运行命令时所在的目录。

如果未指定值，则该命令将访问 Eclipse 缺省工作空间。

将路径嵌入在双引号中。

cmdFile

指定 EGL 命令文件中描述的文件的绝对路径或相对路径。相对路径相对于您运行命令时所在的目录。

将路径嵌入在双引号中。

命令文件必须在工作空间中；否则使用 Eclipse 导入进程来导入该文件，然后重新运行 EGLCMD。

-generateFile *genFile*

指定包含要处理的部件的 EGL 文件的绝对路径或相对路径。相对路径相对于您运行命令时所在的目录。

将路径嵌入在双引号中。

-buildDescriptorFile *bdFile*

指定包含构建描述符的构建文件的绝对路径或相对路径。相对路径相对于您运行命令时所在的目录。

将路径嵌入在双引号中。

-buildDescriptorName *bdName*

指定用来指导生成的构建描述符部件的名称。构建描述符必须位于 EGL 构建文件（.eglbld）的顶层。

-sqlID *sqlID*

设置构建描述符选项 sqlID 的值。

-sqlPassword *sqlPW*

设置构建描述符选项 sqlPassword 的值。

-destUserid *destID*

设置构建描述符选项 destUserID 的值。

-destPassword *destPW*

设置构建描述符选项 destPassword 的值。

您在调用命令 EGLCMD 时指定的构建描述符选项优先于 EGL 命令文件中列示的构建描述符（如果有的话）中的选项。

示例

在下面的命令中，每个多行示例都属于单一行：

```
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
```

相关概念

第 301 页的『从工作台批处理接口生成』

相关任务

第 301 页的『从工作台批处理接口生成』

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 353 页的『destPassword』

第 354 页的『destUserID』

第 364 页的『sqlID』

第 366 页的『sqlPassword』

EGL 命令文件

EGL 命令文件指示当在工作台外部生成输出时要处理的 EGL 文件，无论您是正在使用工作台批处理接口 (EGLCMD 命令) 还是 EGL SDK (EGLSDK 命令)。可以采用下列两种方法的其中一种来创建文件：

- 根据稍后描述的规则通过手工来创建；或者
- 通过使用“EGL 生成”向导来创建，如在工作台中生成中所述。

命令文件是一个 XML 文件，并且文件名必须具有扩展名 .xml，采用大写字母和小写字母的任意组合。文件内容必须符合以下文档类型定义 (DTD)：

```
installationDir\egl\eclipse\plugins\  
com.ibm.etools.egl.utilities_version\  
dtd\eglcommands_5_1.dtd
```

installationDir

产品安装目录，如 C:\Program Files\IBM\RSPD\6.0。如果在安装您要使用的产品之前安装了 Rational Developer 产品并将它保留下来，则可能需要指定在之前安装中使用的目录。

version

插件的已安装版本；例如，6.0.0

下表显示了 DTD 支持的元素和属性。元素和属性名是区分大小写的。

元素	属性	属性值
EGLCOMMANDS (必需)	eglp ath	如 <i>eglp</i> ath 中所述， <i>eglp</i> ath 属性标识当 EGL 使用 <i>import</i> 语句来解析部件名时要搜索的目录。该属性是可选的，如果存在的话，则引用具有一个或多个目录名的加引号字符串，每个目录名之间用分号隔开。 仅当 EGLSDK 命令正在引用命令文件时才使用该属性。如果正在使用 EGLCMD 命令，则会忽略 <i>eglp</i> ath 的值；然而，将根据 EGL 项目路径来解析 <i>import</i> 语句，如 <i>Import</i> 所述。
buildDescriptor (可选；如果您正在使用主构建描述符，则可以避免指定此值，如构建描述符部件中所述)	name	用来指导生成的构建描述符部件的名称。构建描述符必须位于 EGL 构建文件 (.eglbld) 的顶层。 您在调用 EGLCMD 或 EGLSDK 时指定的构建描述符选项优先于 EGL 命令文件中列示的构建描述符（如果有的话）中的选项。
	file	包含构建描述符的 EGL 文件的绝对路径或相对路径。为 EGLCMD 指定的相对路径相对于 Enterprise Developer 工作空间的路径名。为 EGLSDK 指定的相对路径相对于您运行命令时所在的目录。 如果路径包含空格，则必须用双引号将路径引起来。

元素	属性	属性值
generate (可选)	file	<p>包含要处理的部件的 EGL 文件的绝对路径或相对路径。为 EGLCMD 指定的相对路径相对于 Enterprise Developer 工作空间的路径名。为 EGLSDK 指定的相对路径相对于您运行命令时所在的目录。</p> <p>如果路径包含空格，则必须用双引号将路径引起来。</p> <p>如果省略 file 属性，则不会进行生成。</p>

命令文件的示例

本节显示两个命令文件。如果在 EGL 程序文件所在的目录中运行 EGLSDK 命令，则这两个文件生成相同的结果（无论您是使用 EGLCMD 命令还是 EGLSDK 命令）。

以下命令文件包含 generate 命令，该命令使用构建描述符 myBDescPart 来生成程序 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>
```

下一个示例包含两个 generate 命令，这两个命令都隐式地使用主构建描述符。

```
<?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>
```

相关概念

- 第 267 页的『构建描述符部件』
- 第 302 页的『从 EGL 软件开发包 (SDK) 生成』
- 第 301 页的『从工作台批处理接口生成』
- 第 30 页的『Import』

相关任务

- 第 302 页的『从 EGL 软件开发包 (SDK) 生成』
- 第 301 页的『从工作台批处理接口生成』
- 第 298 页的『在工作台中生成』

相关参考

- 第 436 页的『EGLCMD』
- 第 439 页的『EGL 命令文件』
- 第 435 页的『EGL 构建路径和 eglpath』
- 第 446 页的『EGLSDK』

EGL 编辑器

要更改 EGL 文件（扩展名为 `.egl`），请在 EGL 源代码编辑器中工作，该编辑器将通过内容辅助为您提供指导。

要更改 EGL 构建文件（扩展名为 `.eglbld`），执行下列其中一个过程：

- 创建构建文件
-
- 添加构建描述符部件
- 添加链接选项部件
-
- 添加资源关联部件

相关概念

第 13 页的『EGL 项目、包和文件』

第 16 页的『部件』

相关参考

『EGL 中的内容辅助』

EGL 中的内容辅助

EGL 编辑器提供了内容辅助，后者提供可以添加到源文件中的信息建议。只需进行一两次击键，您就可以完成部件、变量或函数的名称，也可以将模板（部件的概要内容）放到源文件中。

用来激活内容辅助的击键是 **Alt + /**。

相关任务

第 109 页的『设置模板首选项』

第 118 页的『将 EGL 模板与内容辅助配合使用』

EGL 中的枚举

在某些情况下，EGL 中的属性或字段的值被限制为特定枚举的值，枚举是一个预定义值类别。例如，属性 `color` 接受枚举 `ColorKind` 的值，并且该枚举的有效值包括 `white` 和 `red`。

可使用枚举名称限定枚举值，以便可将上述值声明为 `ColorKind.white` 和 `ColorKind.red`。但是，仅当代码对与枚举值同名的变量或常量具有访问权时，才需要限定该枚举值。例如，如果作用域中存在名为 `red` 的变量，则符号 `red` 指的是该变量而不是枚举值。

以下枚举列表包括若干枚举值；但这里并没有对这些值进行说明，而是在枚举在其中有意义的属性或字段的上下文中进行了说明：

AlignKind

`center`

`left`

`none`

right

Boolean

yes

no

CallingConventionKind

I4GL

Library

CaseFormatKind

defaultCase

lower

upper

ColorKind

black (仅对控制台字段有效)

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 (仅对控制台字段有效)
word

OutlineKind

bottom
left
right
top

注: sysLib.box 是等同于 [left,right,top,bottom] 的常量。sysLib.noOutline 是一个常量, 表示没有轮廓。

PfKeyKind

pfn, where (1 <= n <=24)

ProtectKind

skip
no
yes

SelectTypeKind

index
value

SignKind

leading
none
parens
trailing

相关概念

第 59 页的『EGL 属性概述』第 53 页的『引用 EGL 中的变量』

EGL 保留字

EGL 包括两种类别的保留字:

- 当您未在处理 SQL 语句时, 为特定用途保留的字
- 当您正在处理 SQL 语句时, 为特定用途保留的字

在 SQL 语句之外的保留字

在 SQL 语句之外, 保留字如下所示, 它们可以是任何大写字母和小写字母的组合:

- 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

在 SQL 语句中的保留字

在 SQL 语句中，保留字如下所示，它们可以是任何大写字母和小写字母的组合：

- 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
- schema, scroll, second, section, select, session, session_user, set, signal, size, smallint, some, space, sql, sqlcode, sqlerror, sqlstate, substr, substring, sum, system_user
- table, tablespace, temporary, terminate, then, time, timestamp, timezone_hour, timezone_minute, tinyint, to, trailing, transaction, translate, translation, trim, true
- uncatalog, union, unique, unknown, update, upper, usage, user, using
- values, varbinary, varchar, varchar2, varying, view
- when, whenever, where, with, work, write
- year
- zone

相关参考

第 80 页的『EGL 语句』

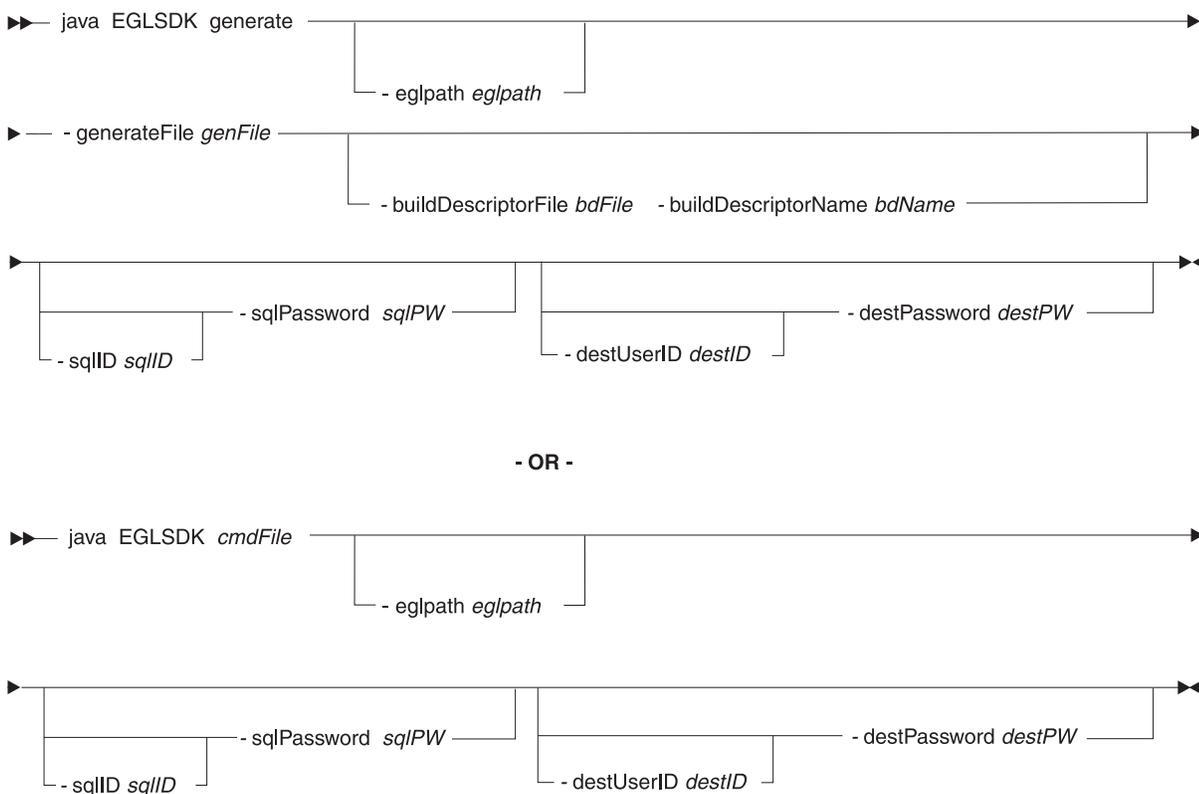
第 612 页的『命名约定』

EGLSDK

命令 EGLSDK 使您能够访问 EGL 软件开发包 (EGL SDK)，如从 *EGL SDK* 生成中所述。

语法

EGLSDK 的调用语法如下所示:



generate

指示该命令本身引用用来生成输出的 EGL 文件和构建描述符部件。在此例中，命令 EGLSDK 未引用命令文件。

cmdFile

指定 EGL 命令文件中描述的文件的绝对路径或相对路径。相对路径相对于您运行命令时所在的目录。

将路径嵌入在双引号中。

-eglpath eglpath

如 *eglpath* 中所述，*eglpath* 选项标识当 EGL 使用 `import` 语句来解析部件名时要搜索的目录。指定一个加引号的字符串，该字符串具有一个或多个目录名，每个目录名之间用分号隔开。

-generateFile *genFile*

包含要处理的部件的 EGL 文件的绝对路径或相对路径。相对路径相对于您运行命令时所在的目录。

将路径嵌入在双引号中。

-buildDescriptorFile *bdFile*

包含构建描述符的构建文件的绝对路径或相对路径。相对路径相对于您运行命令时所在的目录。

将路径嵌入在双引号中。

-buildDescriptorName *bdName*

用来指导生成的构建描述符部件的名称。构建描述符必须位于 EGL 构建文件 (.eglbld) 的顶层。

-sqlID *sqlID*

设置构建描述符选项 sqlID 的值。

-sqlPassword *sqlPW*

设置构建描述符选项 sqlPassword 的值。

-destUserid *destID*

设置构建描述符选项 destUserID 的值。

-destPassword *destPW*

设置构建描述符选项 destPassword 的值。

您在调用命令 EGLSDK 时指定的 `eglpath` 值优先于 EGL 命令文件中的任何 `eglpath` 值。同样，您在调用命令时指定的构建描述符选项优先于 EGL 命令文件中列示的任何构建描述符中的选项。

示例

在下面的命令中，每个多行示例都属于单一行：

```
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
```

相关概念

第 267 页的『构建描述符部件』
第 302 页的『从 EGL 软件开发包 (SDK) 生成』
第 30 页的『Import』
第 270 页的『主构建描述符』

相关任务

第 302 页的『从 EGL 软件开发包 (SDK) 生成』

相关参考

- 第 353 页的『destPassword』
- 第 354 页的『destUserID』
- 第 435 页的『EGL 构建路径和 eglpath』
- 第 364 页的『sqlID』
- 第 366 页的『sqlPassword』
- 第 690 页的『EGL 语句和命令的语法图』

eglmaster.properties 文件的格式

eglmaster.properties 文件是一个 Java 属性文件，EGL SDK 使用它来指定主构建描述符的名称和文件路径名。此属性文件必须包含在调用 EGLSDK 命令的进程的 CLASSPATH 变量所指定的目录中。eglmaster.properties 文件的格式如下所示：

```
masterBuildDescriptorName=desc
masterBuildDescriptorFile=path
```

其中：

desc

主构建描述符的名称

path

EGL 文件的标准路径名，在该文件中声明 EGL SDK 使用的主构建描述符

此文件的内容必须遵循 Java 属性文件的规则。可使用斜杠 (/) 或两个反斜杠 (\) 在路径名中隔开文件名。

必须在属性文件中同时指定 **masterBuildDescriptorName** 和 **masterBuildDescriptorFile** 关键字。否则，eglmaster.properties 文件将被忽略。

以下是 eglmaster.properties 文件内容的示例：

```
# Specify the name of the master build descriptor:
masterBuildDescriptorName=MYBUILDDESRIPTOR
# Specify the file that contains the master build descriptor:
masterBuildDescriptorFile=d:/egl/build descriptors/master.egl
```

相关概念

- 第 270 页的『主构建描述符』

相关任务

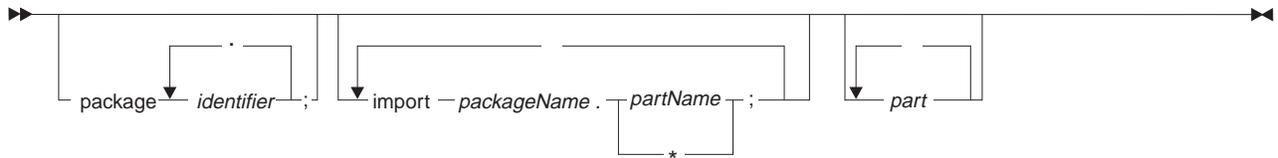
- 第 273 页的『选择 Java 生成选项』

相关参考

- 第 347 页的『构建描述符选项』
- 第 446 页的『EGLSDK』
- 第 463 页的『主构建描述符 plugin.xml 文件的格式』

EGL 源格式

可以在 EGL 源文件中声明逻辑、数据和用户界面部件，每个 EGL 文件都具有 .egl 扩展名并具有以下结构：



package *identifier*

指定文件所在的包的名称，每个标识都通过句点与下一个标识隔开。

有关概述，请参阅 *EGL 项目、包和文件*。

import *packageName*

指定要导入的包的全名。有关概述，请参阅导入。

partName

指定要导入的单个部件。

* 指示要导入包中的每个部件。

part

其中一个 EGL 逻辑、数据或用户界面部件。

可以在 EGL 文件中添加注释，并且既可以在记录内添加注释也可以在记录外添加注释。

相关概念

第 13 页的『EGL 项目、包和文件』

第 30 页的『Import』

第 20 页的『对部件的引用』

第 16 页的『部件』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 345 页的『EGL 源格式的基本记录部件』

第 399 页的『注释』

第 431 页的『EGL 源格式的 DataItem 部件』

第 432 页的『EGL 源格式的 DataTable 部件』

第 464 页的『EGL 源格式的 FormGroup 部件』

第 467 页的『EGL 源格式的表单部件』

第 481 页的『EGL 源格式的函数部件』

第 488 页的『EGL 源格式的带索引记录部件』

第 592 页的『EGL 源格式的库部件』

第 603 页的『EGL 源格式的 MQ 记录部件』

第 619 页的『EGL 源格式的 PageHandler 部件』

第 664 页的『EGL 源格式的程序部件』

第 676 页的『EGL 源格式的相关记录部件』

第 679 页的『EGL 源格式的串行记录部件』

第 683 页的『EGL 源格式的 SQL 记录部件』

EGL 系统异常

EGL 系统异常在整个代码中可用，但通常用于 **onException** 块。有关概述，请参阅[异常处理](#)。

每个 EGL 系统异常至少具有下列字段：

code

一个标识异常的字符串；例如“com.ibm.egl.InvocationException”或等效的常量 `SysLib.InvocationException`

description

一个说明异常含义的字符串

EGL 系统异常如下所示：

SysLib.FileIOException

标识文件访问期间发生的错误。关系数据库的消息队列访问期间发生的错误不会导致此异常。特定于异常的字段如下所示：

errorCode

`SysVar.ErrorCode` 中还会返回 8 个字符的状态码；有关详细信息，请参阅 [SysVar.ErrorCode](#)

fileName

要访问的文件的逻辑名；有关详细信息，请参阅[资源关联和文件类型](#)

SysLib.InvocationException

标识 **call** 语句中发生的错误。

特定于异常的字段如下所示：

errorCode

`SysVar.ErrorCode` 中还会返回 8 个字符的状态码；有关详细信息，请参阅 [SysVar.ErrorCode](#)

name

要调用的程序的名称。

SysLib.LobProcessingException

标识处理类型为 LOB 或 CLOB 的字段期间发生的错误。特定于异常的字段如下所示：

itemName

字段的名称

operation

失败的 EGL 系统函数的名称

resource

连接至该字段的文件（如果有的话）的名称

SysLib.MQIOException

标识访问 MQSeries 消息队列期间发生的错误。特定于异常的字段如下所示：

errorCode

`SysVar.ErrorCode` 中还会返回 8 个字符的状态码；有关详细信息，请参阅 [SysVar.ErrorCode](#)

mqConditionCode

MQSeries API 调用中的完成代码，如 *VGVar.mqConditionCode* 中所述

name

要访问的队列的逻辑名；有关详细信息，请参阅[资源关联和文件类型](#)

SysLib.SQLException

标识关系数据库的访问期间发生的错误。特定于异常的字段如下所示：

sqlca

SQL 通信区；有关详细信息，请参阅 *SysVar.sqlca*

sqlcode

SQL 返回码；有关详细信息，请参阅 *SysVar.sqlcode*

sqlErrd

包含 6 个元素的数组，其中每个元素包含从上一个 SQL I/O 选项返回的相应 SQL 通信区 (SQLCA) 值；有关详细信息，请参阅 *VGVar.sqlErrd*

sqlErrmc

有关通过 JDBC 以外的方式进行的数据库存取的，与 sqlcode 相关联的错误消息；有关详细信息，请参阅 *VGVar.sqlErrmc*

sqlState

最近完成的 SQL I/O 操作的 SQL 状态值；有关详细信息，请参阅 *SysVar.sqlState*

sqlWarn

包含 11 个元素的数组，其中每个元素包含在上一个 SQL I/O 操作的 SQL 通信区 (SQLCA) 中返回的警告字节，并且下标比 SQL SQLCA 描述中的警告编号大 1；有关详细信息，请参阅 *VGVar.sqlState*

相关概念

第 277 页的『[资源关联和文件类型](#)』

相关参考

第 450 页的『[EGL 系统异常](#)』

第 853 页的『[errorCode](#)』

第 857 页的『[sqlca](#)』

第 858 页的『[sqlcode](#)』

第 858 页的『[sqlState](#)』

第 869 页的『[mqConditionCode](#)』

第 870 页的『[sqlerrd](#)』

第 871 页的『[sqlerrmc](#)』

第 872 页的『[sqlWarn](#)』

EGL 系统限制

对于 EGL 文件中的部件数或分层级别数，EGL 定义的限制不起作用。但是，存在下列限制：

- 程序可以使用的变量和文字（包括变量中的字段）的数目不能超过 32767。
- call 语句中的自变量数目不能超过 30 个；同时，自变量的总大小有下列限制：
 - 如果该调用的 **type** 属性值是 remoteCall 或 ejbCall，则不能超过 32567。

上述两个属性都在链接选项部件的 `callLink` 元素中。

- 字段不能超过 32767 个字节。
- 在大多数情况下，数字文字或数字字段的长度包括符号和 / 或小数点在内不能超过 32 位；但对于接收通过调用 `mathLib.round` 函数创建的结果的字段，其长度包括符号和 / 或小数点在内不能超过 31 位。
- 静态数组的维不能超过 7 个，并且元素总数不能超过 32767 个。
- 对于动态数组，情况如下所示：
 - 动态数组的维不能超过 14 个。动态记录数组中的维数是 1（用于记录数组声明）加上记录结构中的维数。
 - 动态数组的最大大小不能超过 2,147,483,647 个元素。如果未指定最大大小，则该数目生效，但在运行时可用的内存容量进一步限制了可以分配的大小。
 - 进行远程调用时可以传递的所有自变量的总大小受协议所支持的最大缓冲区大小的限制。

相关参考

第 370 页的『`callLink` 元素』

第 781 页的『`round()`』

第 612 页的『命名约定』

第 378 页的『`callLink` 元素中的 `parmForm`』

第 385 页的『`callLink` 元素中的 `type`』

表达式

表达式是您在编写程序或函数脚本时指定的一系列操作数和运算符。

在运行时，每个表达式都解析为特定类型的值。数字表达式解析为数字；字符串表达式解析为一系列字符；逻辑表达式解析为 `true` 或 `false`；日期时间表达式解析为日期、时间间隔、时间或时间戳记。

表达式将按一组优先顺序规则和（在给定级别的优先顺序中）从左至右的顺序求值，但您可以使用圆括号来强制使用不同的顺序。先对嵌套插入子表达式求值，然后再对其外部的插入子表达式求值，并且所有插入表达式在整个表达式之前求值。

在给定级别的求值中，第一个操作数确定表达式（或子表达式）的类型。请参照以下示例：

```
"A value = " + 1 + 2
```

第一个操作数是字符类型，表达式是带有以下值的文本表达式：

```
"A value = 12"
```

考虑另一文本表达式：

```
"A value = " + (1 + 2)
```

此情况下的值如下：

```
"A value = 3"
```

相关参考

第 453 页的『日期时间表达式』

第 454 页的『逻辑表达式』

第 461 页的『数字表达式』

第 462 页的『文本表达式』

日期时间表达式

根据上下文，日期时间表达式解析为类型为 DATE、INT、INTERVAL、TIME 或 TIMESTAMP 的值。日期时间表达式必须包括下列其中一项：

- 包含其中一个类型的值的变量。
- 返回日期时间值的函数调用。若干系统函数从字符串文字或常量创建日期时间值：
 - **DateTimeLib.dateValue** 创建日期
 - **DateTimeLib.intervalValue** 创建时间间隔
 - **DateTimeLib.timeValue** 创建时间
 - **DateTimeLib.timeStampValue** 创建时间戳记

系统函数 **DateTimeLib.extend** 还会返回长度比类型为 DATE、TIME 或 TIMESTAMP 的输入字段更长或短的时间戳记值。

下表总结在日期时间表达式中有效的算术运算的类型。就像显示的那样，日期时间表达式可能包括返回数字的数字表达式，但仅在少数情况下才会如此。

日期时间表达式中的算术运算

操作数 1 的类型	运算符	操作数 2 的类型	结果类型	注释
DATE	-	DATE	INT	
DATE	+/-	NUMBER	DATE	
NUMBER	+	DATE	DATE	
TIME STAMP	-	TIMESTAMP	INTERVAL	INTERVAL(dd, ss) 除非操作数 1 和操作数 2 为下列任何一项： <ul style="list-style-type: none">• TIMESTAMP(yyyy)• TIMESTAMP(yyyyMM)• TIMESTAMP(MM) 在这三种情况下，结果为 INTERVAL(yyyyMM)
DATE	-	TIMESTAMP	INTERVAL	INTERVAL(ddssmmffffff)
TIME STAMP	-	DATE	INTERVAL	INTERVAL(ddHHmmssffffff)
TIME STAMP	+/-	INTERVAL	TIMESTAMP	
INTERVAL	+	TIMESTAMP	TIMESTAMP	
DATE	+/-	INTERVAL	TIMESTAMP	
INTERVAL	+	DATE	TIMESTAMP	

日期时间表达式中的算术运算

操作数 1 的类型	运算符	操作数 2 的类型	结果类型	注释
INTERVAL	+/-	INTERVAL	INTERVAL	操作数 1 和操作数 2 必须同时（最多）具有年份和月份或者必须同时（最多）具有日期和时间值
INTERVAL	*//	NUMBER	INTERVAL	

相关参考

第 340 页的『赋值』

第 726 页的『dateValue()』

第 728 页的『extend()』

第 728 页的『intervalValue()』

第 732 页的『timeValue()』

第 731 页的『timeStampValue()』

第 452 页的『表达式』

『逻辑表达式』

第 461 页的『数字表达式』

第 613 页的『运算符和优先顺序』

第 31 页的『基本类型』

第 462 页的『文本表达式』

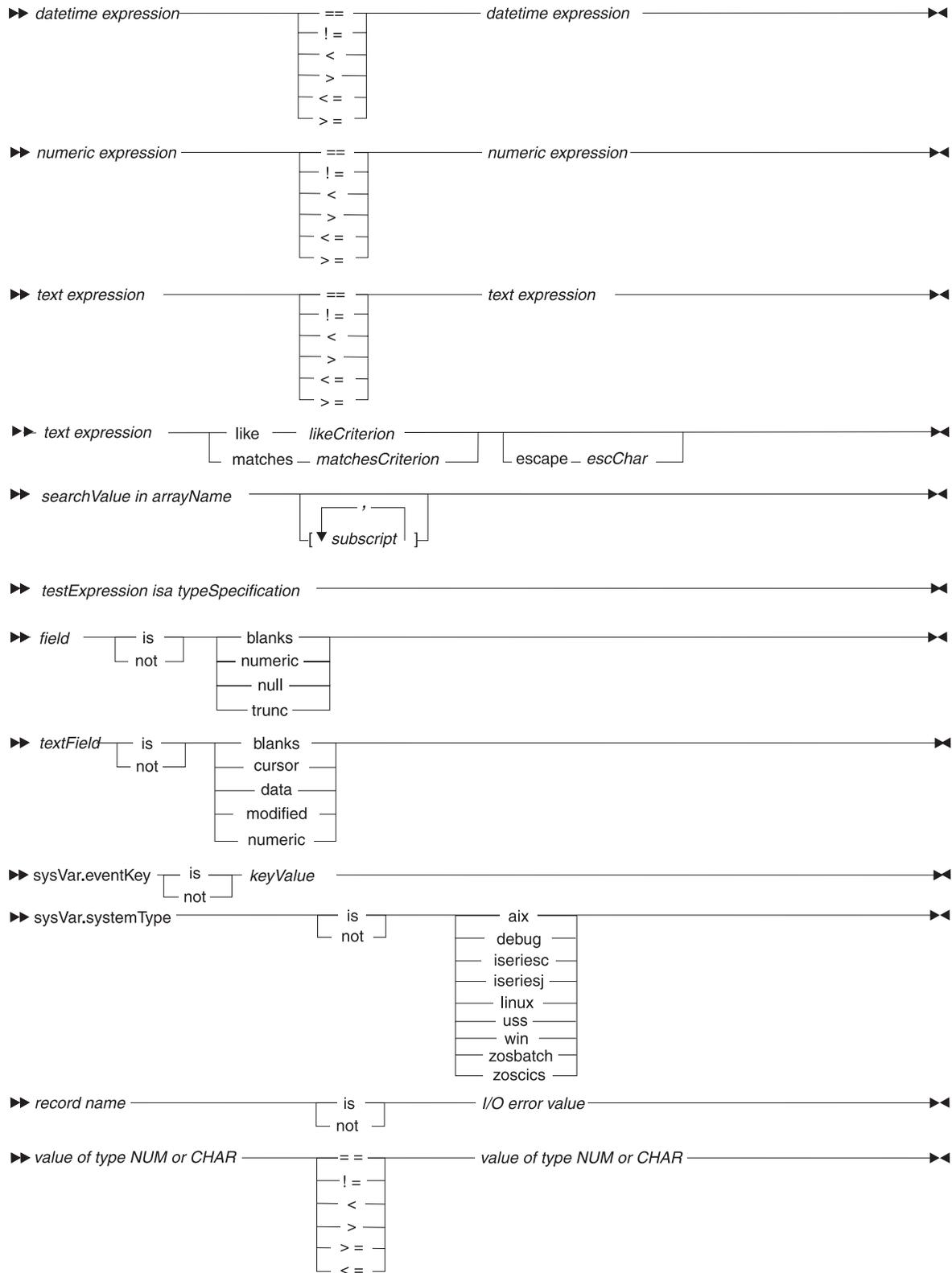
第 688 页的『子串』

逻辑表达式

逻辑表达式的结果为 true 或 false，并且在 **if** 或 **while** 语句或（在某些情况下）**case** 语句中用作条件。

基本逻辑表达式

基本逻辑表达式由操作数、比较运算符和第二个操作数组成，如以下语法图及后续表所示：



第一个操作数	比较运算符	第二个操作数
日期时间表达式	下列其中一项: ==, !=, <, >, <=, >=	日期时间表达式 第一个和第二个表达式必须是兼容类型。 如果是日期时间比较, 则大于号 (>) 表示以后的时间, 小于号 (<) 表示以前的时间。
数字表达式	下列其中一项: ==, !=, <, >, <=, >=	数字表达式
字符串表达式	下列其中一项: ==, !=, <, >, <=, >=	字符串表达式
字符串表达式	like	<i>likeCriterion</i> , 这是将对其比较字符串表达式的字符字段或文字, 比较方式是从左至右比较逐个字符位置。此功能的用法类似于 SQL 查询中的关键字 like 的用法。 <i>escChar</i> 是解析为转义字符的单字符字段或文字。 有关更多详细信息, 请参阅 <i>like</i> 运算符。
字符串表达式	匹配	<i>matchCriterion</i> , 这是将对其比较字符串表达式的字符字段或文字, 比较方式是从左至右比较逐个字符位置。此功能的用法类似于 UNIX 或 Perl 中的正则表达式的用法。 <i>escChar</i> 是解析为转义字符的单字符字段或文字。 有关更多详细信息, 请参阅 <i>matches</i> 运算符。
<i>Value of type NUM or CHAR</i> , 如对第二个操作数所述	下列其中一项: ==, !=, <, >, <=, >=	<i>Value of type NUM or CHAR</i> , 它可以是下列任何一项: <ul style="list-style-type: none">• 类型为 NUM 并且没有小数位的字段• 整数文字• 类型为 CHAR 的字段或文字
<i>searchValue</i>	in	<i>arrayName</i> ; 有关详细信息, 请参阅 <i>in</i> 。
<i>field not in SQL record</i>	下列其中一项: <ul style="list-style-type: none">• is• not	下列其中一项: <ul style="list-style-type: none">• blanks (用于测试字符字段的值是否只包含空格)• numeric (用于测试类型为 CHAR 或 MBCHAR 的字段的价值是否为数字)

第一个操作数	比较运算符	第二个操作数
<i>field in an SQL record</i>	下列其中一项: <ul style="list-style-type: none"> • is • not 	下列其中一项: <ul style="list-style-type: none"> • blanks (用于测试字符字段的值是否只包含空格) • null (用于测试是否已通过 set 语句或通过从关系数据库读取将该字段设置为 NULL) • numeric (用于测试类型为 CHAR 或 MBCHAR 的字段是否为数字) • trunc (用于测试上一次从关系数据库将单字节或双字节字符值读取到该字段中时, 是否删除右边的非空格字符) <p>仅当数据库列的长度超过该字段的长度时, trunc 测试结果才是 true。在将值移动到字段之后或在将字段设置为 NULL 后, 测试结果值为 false。</p>
<i>textField</i> (文本表单中的字段的名称)	下列其中一项: <ul style="list-style-type: none"> • is • not 	下列其中一项: <ul style="list-style-type: none"> • blanks (用于测试文本字段的值是否被限制为空白或 NULL)。 <p>blanks 测试基于用户在表单中上次输入的内容, 而不是基于表单字段的当前内容; 在下列情况下, 使用 is 的测试将得到 true 结果:</p> <ul style="list-style-type: none"> - 用户上次输入的内容是空白或 NULL; 或者 - 在启动程序之后, 或者在运行具有 <i>set form initial</i> 类型的 set 语句之后, 用户未在字段中输入任何数据。 <ul style="list-style-type: none"> • cursor (用于测试用户是否将光标留在指定的文本字段中)。 • data (用于测试指定的文本字段是否包含除空白或 NULL 以外的数据)。 • modified (用于测试是否已设置字段的已修正数据标记, 如已修正数据标记和 modified 属性所述)。 • numeric (用于测试类型为 CHAR 或 MBCHAR 的字段是否为数字)。
<i>ConverseVar.eventKey</i>	下列其中一项: <ul style="list-style-type: none"> • is • not 	有关详细信息, 请参阅 <i>ConverseVar.eventKey</i> 。
<i>sysVar.systemType</i>	下列其中一项: <ul style="list-style-type: none"> • is • not 	有关详细信息, 请参阅 <i>sysVar.systemType</i> 。 不能使用 <i>is</i> 或 <i>not</i> 来测试 <i>VGLib.getVAGSysType</i> 返回的值。

第一个操作数	比较运算符	第二个操作数
<i>record name</i>	下列其中一项: <ul style="list-style-type: none"> • <i>is</i> • <i>not</i> 	适用于记录组织的 I/O 错误值。请参阅 <i>I/O 错误值</i> 。

下表列示比较运算符，其中每个运算符在表达式中使用时将解析为 true 或 false。

运算符	用途
<code>==</code>	等于运算符指示两个操作数的值是否相等。
<code>!=</code>	不等于运算符指示两个操作数是否具有不同的值。
<code><</code>	小于运算符指示第一个操作数在数字上是否小于第二个操作数。
<code>></code>	大于运算符指示第一个操作数在数字上是否大于第二个操作数。
<code><=</code>	小于等于运算符指示第一个操作数在数字上是否小于等于第二个操作数。
<code>>=</code>	大于等于运算符指示第一个操作数在数字上是否大于等于第二个操作数。
<code>in</code>	<i>in</i> 运算符指示两个操作数中的第一个操作数是否是第二个操作数（它引用数组）中的值。有关详细信息，请参阅 <i>in</i> 。
<code>is</code>	<i>is</i> 运算符指示两个操作数中的第一个操作数是否在第二个操作数的类别中。有关详细信息，请参阅上表。
<code>like</code>	<i>like</i> 运算符指示两个操作数中的第一个操作数中的字符是否与第二个操作数中的字符相匹配，如 <i>like</i> 运算符中所述。
匹配	<i>matches</i> 运算符指示两个操作数中的第一个操作数中的字符是否与第二个操作数中的字符相匹配，如 <i>matches</i> 运算符中所述。
<code>not</code>	<i>not</i> 运算符指示两个操作数中的第一个操作数是否不在第二个操作数的类别中。有关详细信息，请参阅上表。

下表及随后的解释说明操作数具有指定类型时的兼容性规则。

第一个操作数的基本类型	第二个操作数的基本类型
BIN	BIN, DECIMAL, FLOAT, MONEY, NUM, NUMC, PACF, SMALLFLOAT
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

第一个操作数的基本类型	第二个操作数的基本类型
PACF	BIN, DECIMAL, FLOAT, MONEY, NUM, NUMC, PACF, SMALLFLOAT
TIME	CHAR, NUM, TIME, TIMESTAMP
TIMESTAMP	CHAR, DATE, TIME, TIMESTAMP
UNICODE	UNICODE

详细信息如下所示:

- 可将任何数字类型 (BIN、DECIMAL、FLOAT、MONEY、NUM、NUMC、PACF 和 SMALLFLOAT) 的值与任何数字类型和大小的值比较, 并且 EGL 将进行适当的临时转换。相等的小数的等同性比较 (如 1.4 和 1.40) 求值为 true, 即使小数位不同。
- 仅当类型为 CHAR 的每个字符都在十六进制数字 (0 至 9、A 至 F 以及 a 至 f) 的范围内时, 类型为 CHAR 的值才能与类型为 HEX 的值进行比较。在类型为 CHAR 的值中, EGL 将所有小写字母临时转换为大写字母。
- 如果比较包括两个具有字符类型 (CHAR、DBCHAR、HEX、MBCHAR 和 UNICODE) 的值, 并且一个值比另一个值的字节数要少, 则临时转换将在较短值的右边进行填充:
 - 在与类型为 MBCHAR 的值的比较中, 将在类型为 CHAR 的值的右边填充单字节空格
 - 在与类型为 HEX 的值的比较中, 将在类型为 CHAR 的值的右边填充二进制零
 - 在类型为 DBCHAR 的值的右边填充双字节空格
 - 在类型为 UNICODE 的值的右边填充 Unicode 双字节空格
 - 在类型为 HEX 的值的右边填充二进制零, 这意味着 (例如) 如果必须将值“0A”扩展为两个字节, 则用于比较的值为“0A00”而不是“000A”
- 仅当下列条件适用时, 类型为 CHAR 值才能与类型为 NUM 的值进行比较:
 - 类型为 CHAR 的值具有单字节数字, 没有其它任何字符
 - 类型为 NUM 的值的定义没有小数点

CHAR 至 NUM 的比较按如下方式进行:

- 临时转换将 NUM 值放置到 CHAR 格式中。数字字符是向左对齐的, 并根据需要带上附加单字节空格。例如, 如果长度为 4 的 NUM 类型的字段的值为 7, 则将该值视为“7”, 且右边带有三个空格。
- 如果各个字段的长度不匹配, 则临时转换将在较短值的右边填充空格。
- 比较将逐字节地检查值。考虑下列两个示例:
 - 长度为 2、值为“7 ” (包含空格) 并且类型为 CHAR 的字段与长度为 1、值为 7 并且类型为 NUM 的字段相等, 因为基于 NUM 类型的字段的临时字段也包含最后空格
 - 值为“8”的类型为 CHAR 的字段大于值为 534 的类型为 NUM 的字段, 这是因为在 ASCII 或 EBCDIC 搜索顺序中, “8”位于“5”之后

复杂逻辑表达式

通过使用 *and* (&&) 或 *or* 运算符 (||) 组合一对较基本的表达式, 可以构建更复杂的表达式。此外, 也可以使用 *not* 运算符 (!), 如稍后所描述的那样。

如果逻辑表达式由 *or* 运算符组合的基本逻辑表达式组成，则 EGL 将根据优先顺序规则对表达式求值，但如果其中一个基本逻辑表达式的结果为 *true*，则将停止求值。请参照以下示例：

```
field01 == field02 || 3 in array03 || x == y
```

如果 *field01* 不等于 *field02*，则求值将继续。但是，如果 *array03* 包含值 3，则整个表达式求值为 *true*，并且不会对最后一个基本逻辑表达式 (*x == y*) 求值。

同样，如果用 *and* 运算符将基本逻辑表达式组合起来，则如果其中一个基本逻辑表达式解析为 *false*，EGL 将停止求值。在以下示例中，只要发现 *field01* 不等于 *field02*，求值就会停止：

```
field01 == field02 && 3 in array03 && x == y
```

可以在逻辑表达式中使用成对圆括号来实现下列任何目的：

- 更改求值顺序。
- 阐明意义。
- 能够使用 *not* 运算符 (!)，它解析为与其后紧跟的逻辑表达式的值相反的布尔值 (*true* 或 *false*)。必须将后继表达式括在圆括号中。

示例

在查看下面的示例时，假定 *value1* 包含“1”，*value2* 包含“2”，依此类推：

```
/* == true */
value5 < value2 + value4

/* == false */
!(value1 is numeric)

/* == true when the generated output runs
   on Windows 2000, Windows NT,
   or z/OS UNIX System Services */
sysVar.systemType is WIN || sysVar.systemType is USS

/* == true */
(value6 < 5 || value2 + 3 >= value5) && value2 == 2
```

相关概念

第 148 页的『已修正数据标记和 *modified* 属性』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 515 页的『*case*』

第 453 页的『日期时间表达式』

第 86 页的『异常处理』

第 452 页的『表达式』

第 490 页的『I/O 错误值』

第 555 页的『*if, else*』

第 486 页的『*in* 运算符』

第 597 页的『*like* 运算符』

第 597 页的『*like* 运算符』

- 『数字表达式』
- 第 613 页的『运算符和优先顺序』
- 第 31 页的『基本类型』
- 第 462 页的『文本表达式』
- 第 845 页的『eventKey』
- 第 842 页的『getVAGSysType()』
- 第 859 页的『systemType』
- 第 591 页的『while』

数字表达式

数字表达式解析为一个数字，可以在各种情况下指定这样的表达式；例如，可以在赋值语句的右边指定数字表达式。数字表达式可以由下列任何内容组成：

- 数字操作数，它是下列其中一项：
 - 包含数字的变量。该项前面可以有一个符号。
 - 数字文字，它可以由符号开头，但总是包含一系列数字，并且可以包含一个小数点。
 - 返回数字的函数调用。

数字文字的类型由该文字的值暗示：

- 4 位或少于 4 位的整数的类型为 SMALLINT
- 5 到 8 位的整数的类型为 INT
- 9 到 18 位的整数的类型为 BIGINT
- 包括小数点的数字的类型为 NUM
- 数字操作数，后跟一个数字运算符，再后跟另一个数字操作数。
- 通过使用数字运算符来组合一对较基本的表达式，形成较复杂的表达式。

可以在数字表达式中使用成对的圆括号来更改求值顺序或明确其含义。

在查看以下示例时，假定 intValue1 等于 1，intValue2 等于 2，依此类推，并且每个值都没有小数位：

```

/* == -8, with the parentheses overriding
      the usual precedence of * and + */
intValue2 * (intValue1 - 5)

/* == -2, with a unary minus as the last operator */
intValue2 + -4

/* == 1.4, if the expression is assigned to an
      item with at least one decimal place. */
intValue7 / intValue5

/* == 2, which is a remainder
      expressed as an integer value */
intValue7 % intValue5

```

有关显示圆括号对加号 (+) 用法的影响的示例，请参阅表达式。

如果计算出来的中间值超过 128 位，则数字表达式可能会给出意外的结果。

相关参考

- 第 453 页的『日期时间表达式』

- 第 452 页的『表达式』
- 第 454 页的『逻辑表达式』
- 第 613 页的『运算符和优先顺序』
- 第 31 页的『基本类型』
- 『文本表达式』

文本表达式

文本表达式解析为一系列字符，可以在各种情况下（例如，在赋值语句的右边）指定这样的表达式。文本表达式可以是下列其中一项：

- 包含一系列字符的变量。
- 字符串文字，它是用双引号定界的一系列字符。类型为 `STRING` 的文字。
- 包含一系列字符的文字或变量的子串。有关详细信息，请参阅子串。
- 返回一系列字符的任何字符串格式的系统字的调用。有关详细信息，请参阅字符串格式（系统字）。
- 先前种类的一系列值，其中值与值之间是用并置运算符（即加号（+））隔开的。下列语句将 `WebSphere` 赋予 `myString`：

```
myString = "Web" + "Sphere";
```

有关显示圆括号对加号（+）用法的影响的示例，请参阅表达式。

- 返回一系列字符的任何其它函数调用。

前面有转义字符（\）的任何字符都包括在文本表达式中。特别是，可以使用转义字符来将下列字符包括在文字、字段或返回值中：

- 双引号（"）
- 反斜杠（\）
- 退格，由 `\b` 指示
- 换页，由 `\f` 指示
- 换行符，由 `\n` 指示
- 回车符，由 `\r` 指示
- 制表符，由 `\t` 指示

示例如下所示：

```
myString = "He said, \"Escape while you can!\"";  
myString2 = "Is a backslash (\\) needed?";
```

如果文字没有结束引号，将发生错误：

```
myString3 = "Escape is impossible\"";
```

文本表达式中的每个值必须对使用该表达式的上下文有效。例如，不能在对类型为 `CHAR` 的项赋值的表达式中使用类型为 `UNICODE` 的项。其它详细信息在赋值中。

相关参考

- 第 340 页的『赋值』
- 第 453 页的『日期时间表达式』
- 第 452 页的『表达式』
- 第 454 页的『逻辑表达式』

第 461 页的『数字表达式』
第 613 页的『运算符和优先顺序』
第 31 页的『基本类型』

第 688 页的『子串』

主构建描述符 `plugin.xml` 文件的格式

主构建描述符 `plugin.xml` 文件是一个 XML 文件，工作台使用它来指定主构建描述符的名称和文件路径名。仅当您需要主构建描述符来强制对生成使用某些选项且正在从工作台生成或正在使用 `EGLCMD` 命令时，才需要此文件。必须将此 `plugin.xml` 文件放置在 `plugins` 目录中的某个目录中。该文件的格式如下所示：

```
<?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>
```

其中：

id 插件的标识

plg
插件的名称

com
公司的名称

bfil
包含主构建描述符的文件的路径名，格式为 *project/folder/file*（相对于 Enterprise Developer 的 `workspace` 目录），其中：

project
项目目录的名称

folder
项目目录中的某个目录的名称

file
包含主构建描述符的文件名称

mas
主构建描述符的名称

此文件的内容必须遵循 XML 文件的规则。要在路径名中隔开文件名，必须使用斜杠（*/*）字符。

必须同时指定名称属性和文件属性。否则 `plugin.xml` 文件会被忽略。

以下是 `plugin.xml` 文件的内容的示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Example master BuildDescriptor Plugin -->
```

```

<plugin
  id="example.master.BuildDescriptor.plugin"
  name="Example master BuildDescriptor plug-in"
  version="5.0"
  vendor-name="IBM">
  <requires />
  <runtime />
  <!-- ===== -->
  <!-- -->
  <!-- Register the master BuildDescriptor -->
  <!-- -->
  <!-- ===== -->
  <extension point =
    "com.ibm.etools.egl.generation.base.framework.masterBuildDescriptor" >
    <masterBuildDescriptor file
      = "myProject/myFolder/myFile.eglbuild" name = "masterBD" />
    </extension>
</plugin>

```

相关概念

第 267 页的『构建描述符部件』

第 270 页的『主构建描述符』

相关任务

第 301 页的『从工作台批处理接口生成』

第 298 页的『在工作台中生成』

相关参考

第 347 页的『构建描述符选项』

第 436 页的『EGLCMD』

第 448 页的『eglmaster.properties 文件的格式』

EGL 源格式的 FormGroup 部件

可以在 EGL 文件中声明 formGroup 部件，EGL 源格式对该部件作了描述。这个部件是主部件，这意味着它必须位于文件的顶层，并且必须与文件同名。

程序只能使用与程序的使用声明所引用的表单组相关联的表单。

下面是 formGroup 部件的一个示例：

```

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
  }
}

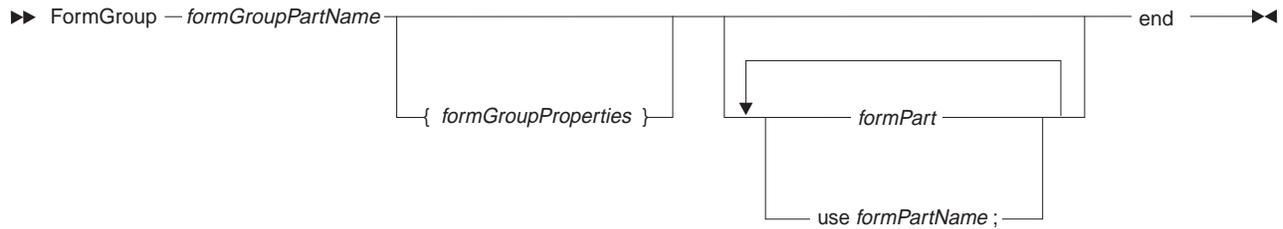
```

```

    }
  }
  use myForm01;
  use myForm02;
end

```

formGroup 部件的图如下所示:



FormGroup *formGroupName* ... end

将部件标识为表单组并指定部件名。有关命名规则，请参阅命名约定。

formGroupProperties

一系列属性，用逗号将每个属性与下一个属性隔开。下面的内容对每个属性都作了描述。

formPart

文本或打印表单，如 *EGL* 源格式的表单部件中所述。

use *formPartName*

使用声明，它提供对未嵌入表单组的表单的访问权。

表单组属性如下所示:

alias

一个字符串，它包含在生成的输出的名称中。如果未指定别名，则会使用 `FormGroup` 部件名。

validationBypassKeys = [*bypassKeyValue*]

标识一个或多个用户击键，该击键导致 *EGL* 运行时跳过输入字段验证。此属性对于保留用于快速结束程序的击键而言很有用。每个 *bypassKeyValue* 选项如下所示:

pf*n*

F 或 PF 键的名称，包括介于 1 与 24 之间（包括 1 和 24）的数字。

注: PC 键盘上的功能键通常是 *F* 键，如 F1，但 *EGL* 使用 IBM *PF* 术语，所以 F1（例如）被称为 PF1。

如果您希望指定多个键值，则使用方括号来对一组值进行定界，并用逗号将相邻的两个值隔开，如以下示例所示:

```
validationBypassKeys = [pf3, pf4]
```

helpKey = "*helpKeyValue*"

标识一个用户击键，该击键使 *EGL* 运行时向用户显示帮助表单。*helpKeyValue* 选项如下所示:

f*n*

f 或 pf 键的名称，包括介于 1 与 24 之间（包括 1 和 24）的数字。

注: PC 键盘上的功能键通常是 *f* 键, 如 f1, 但 EGL 使用 IBM *pf* 术语, 所以 f1 (例如) 被称为 pf1。

pfKeyEquate = yes, pfKeyEquate = no

指定当用户按下大编号功能键 (PF13 至 PF24) 时注册的击键是否与用户按下编号小于 12 的功能键时注册的击键相同。有关详细信息, 请参阅 *pfKeyEquate*。

screenFloatingArea { properties }

定义用于输出到屏幕的浮动区域。有关浮动区域的概述, 请参阅 *表单部件*。有关属性详细信息, 请参阅下一节。

printFloatingArea { properties }

定义用于可打印输出的浮动区域。有关浮动区域的概述, 请参阅 *表单部件*。有关属性详细信息, 请参阅 *打印浮动区域的属性*。

屏幕浮动区域的属性

screenFloatingArea 后面的属性集由花括号 ({ }) 定界, 并且用逗号将每个属性与下一个属性隔开。属性如下所示:

screenSize = [rows, columns]

联机显示区域中的行数和列数, 包括用作页边空白的任何行或列。缺省值如下所示:

```
screenSize=[24,80]
```

topMargin= rows

在显示区域顶部留下的空行数目。缺省值为 0。

bottomMargin= rows

在显示区域底部留下的空行数目。缺省值为 0。

leftMargin= columns

在显示区域左边留下的空列数目。缺省值为 0。

rightMargin= columns

在显示区域右边留下的空列数目。缺省值为 0。

打印浮动区域的属性

printFloatingArea 后面的属性集由花括号 ({ }) 定界, 并且用逗号将每个属性与下一个属性隔开。属性如下所示:

pageSize = [rows, columns]

可打印显示区域中的行数和列数, 包括用作页边空白的任何行或列。如果指定了打印浮动区域, 则此属性是必需的。

deviceType = singleByte, deviceType = doubleByte

指定浮动区域声明是用于支持单字节输出的打印机 (缺省情况) 还是用于支持双字节输出的打印机。如果任何表单包含类型为 DBCHAR 或 MBCHAR 的项, 则指定 **doubleByte**。

topMargin = rows

在显示区域顶部留下的空行数目。缺省值为 0。

bottomMargin = rows

在显示区域底部留下的空行数目。缺省值为 0。

leftMargin = columns

在显示区域左边留下的空列数目。缺省值为 0。

rightMargin = columns

在显示区域右边留下的空列数目。缺省值为 0。

相关概念

第 13 页的『EGL 项目、包和文件』

第 141 页的『FormGroup 部件』

第 142 页的『表单部件』

相关参考

第 448 页的『EGL 源格式』

『EGL 源格式的表单部件』

第 612 页的『命名约定』

第 625 页的『pfKeyEquate』

第 875 页的『使用声明』

EGL 源格式的表单部件

可以在 EGL 文件中声明表单部件，*EGL* 源格式对该部件作了描述。如果表单部件仅由一个表单组访问，则建议将该表单部件嵌入在 `formGroup` 部件中。如果表单部件由多个表单组访问，则有必要在 EGL 文件的最上层指定该表单部件。

以下是文本表单的一个示例：

```
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
```

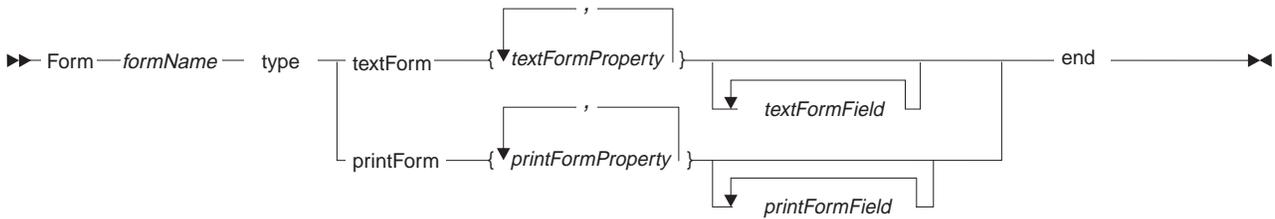
以下是打印表单的一个示例：

```

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

```

表单部件的图如下所示:



Form formName ... end

将部件标识为表单并指定部件名。有关命名规则，请参阅命名约定。

textForm

指示该表单为文本表单。

textFormProperty

一个文本表单属性。有关详细信息，请参阅文本表单。

textFormField

一个文本表单字段。有关详细信息，请参阅表单字段。

printForm

指示该表单为打印表单。

printFormProperty

一个打印表单属性。有关详细信息，请参阅打印表单。

printFormField

一个打印表单字段。有关详细信息，请参阅表单字段。

文本表单属性

文本表单属性如下所示:

formSize = [rows, columns]

联机显示区域中的行数和列数。此属性是必需的。

列值与可以在显示区域中横向显示的单字节字符数相等。

position = [row, column]

表单在显示区域中的显示位置的行和列。如果省略此属性，则该表单为浮动表单并显示在浮动区域中，显示位置为能够放得下整个表单的浮动区域中的下一可用行。

validationBypassKeys = [bypassKeyValue]

标识一个或多个用户击键，该击键导致 EGL 运行时跳过输入字段验证。此属性对于保留用于快速结束程序的击键而言很有用。bypassKeyValue 选项如下所示:

pf*n*

F 或 PF 键的名称，包括介于 1 与 24 之间（包括 1 和 24）的数字

注：PC 键盘上的功能键通常是 *F* 键，如 F1，但 EGL 使用 IBM *PF* 术语，所以 F1（例如）被称为 PF1。

如果您希望指定多个键值，则使用圆括号来对一组值进行定界，并用逗号将相邻的两个值隔开，如下示例所示：

```
validationBypassKeys = [pf3, pf4]
```

helpKey = "helpKeyValue"

标识一个用户击键，该击键使 EGL 运行时向用户显示帮助表单。*helpKeyValue* 选项如下所示：

pf*n*

F 或 PF 键的名称，包括介于 1 与 24 之间（包括 1 和 24）的数字

注：PC 键盘上的功能键通常是 *F* 键，如 F1，但 EGL 使用 IBM *PF* 术语，所以 F1（例如）被称为 PF1。

helpForm = "formName"

特定于文本表单的帮助表单的名称。

msgField = "fieldName"

文本表单字段的名称，该字段显示一条消息以作为对验证错误的响应或对运行 `ConverseLib.displayMsgNum` 的响应。

alias = "alias"

不超过 8 个字符的别名，供 EGL 运行时使用。

打印表单属性

打印表单属性如下所示：

formsize = [rows, columns]

联机显示区域中的行数和列数。此属性是必需的。

列值与可以在显示区域中横向显示的单字节字符数相等。

position = [row. column]

表单在显示区域中的显示位置的行和列。如果省略此属性，则该表单为浮动表单并显示在浮动区域中，显示位置为能够放得下整个表单的浮动区域中的下一可用行。

msgField = "fieldName"

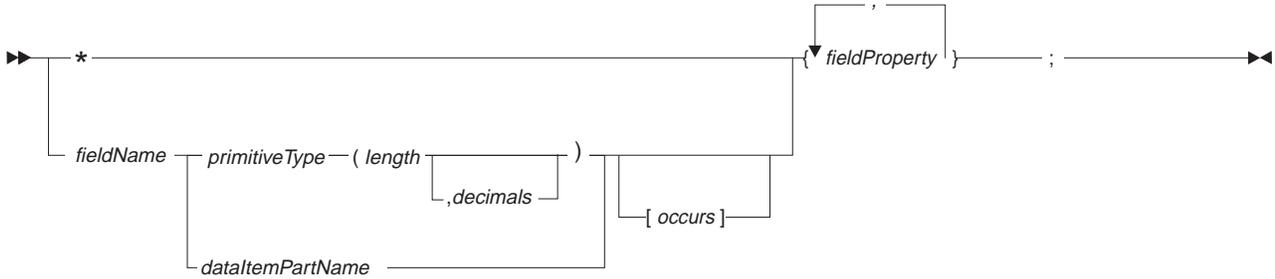
文本表单字段的名称，该字段显示一条消息以作为对运行 `ConverseLib.displayMsgNum` 的响应。

alias = "alias"

不超过 8 个字符的别名，供 EGL 运行时使用。

表单字段

表单字段的图如下所示：



* 指示该字段为常量字段。它没有名称，但具有常量值，该值是在特定于字段的 **value** 属性中指定的。代码中的语句无法访问常量字段中的值。

fieldProperty

一个文本表单字段属性。有关详细信息，请参阅文本表单字段属性。

fieldName

指定字段的名称。有关规则，请参阅命名约定。

代码可以访问已命名字段的值，该字段也被称为变量字段。

如果文本表单包含变量字段，而该变量字段开始于一行并结束于另一行，则只能在屏幕宽度与表单宽度相同的屏幕上显示该文本表单。

occurs

字段数组中的元素数目。仅支持一维数组。有关更多详细信息，请参阅关于字段数组。

primitiveType

对字段指定的基本类型。该指定值影响最大长度；但任何数字字段都可以作为 NUM 类型的字段生成。

包含 DBCHAR 类型的字段的表单只能在支持双字节字符集的系统和设备上使用。同样，包含 MBCHAR 类型的字段的表单只能在支持多字节字符集的系统和设备上使用。

文本或打印表单不支持基本类型 FLOAT、SMALLFLOAT 和 UNICODE。

length

字段的长度，它是一个整数，表示可以放入字段的最大字符或数字的数量。

decimals

对于数字类型（BIN、DECIMAL、NUM、NUMC 或 PACF），可以指定 *decimals*，它是用来表示小数点后的位数的整数。最大小数位数是以下两个数字中较小的那一个：18 或声明为 *length* 的位数。小数点不与数据存储在一起。

dataItemPartName

dataItem 部件的名称，该部件是字段的格式模型，如 *typeDef* 所述。dataItem 部件必须对表单部件可视，如对部件的引用中所述。

文本表单字段属性

只在文本表单字段中才有用的属性以后讨论。下列属性的使用范围较广并且可用：

- 第 629 页的『align』
- 第 632 页的『currency』
- 第 632 页的『currencySymbol』

- 第 633 页的 『 dateFormat 』
- 第 637 页的 『 fillCharacter 』
- 第 639 页的 『 isBoolean 』
- 第 642 页的 『 lineWrap 』
- 第 642 页的 『 lowerCase 』
- 第 642 页的 『 masked 』
- 第 646 页的 『 numericSeparator 』
- 第 646 页的 『 outline 』
- 第 650 页的 『 sign 』
- 第 652 页的 『 timeFormat 』
- 第 653 页的 『 timeStampFormat 』
- 第 654 页的 『 upperCase 』
- 第 660 页的 『 zeroFormat 』

对于任何字段

下列属性对表单上的任何字段都有用:

position = [row. column]

在字段之前的属性字节的行和列。此属性是必需的。

value = "stringLiteral"

在字段中显示的字符串。引号是必需的。

可以对任何项指定此属性; 例如, 在 `dataItem` 部件声明中指定此属性。

注: 如果 VisualAge Generator 兼容性已生效, 并且设置了文本表单属性 **value**, 则仅当用户返回表单之后才能在程序中使用该属性的内容。因此, 在程序中设置的值不需要对程序中的项有效。

fieldLen = lengthInBytes

字段长度; 可以在字段中显示的单字节字符数。此值不包括前导属性字节。

数字字段的 **fieldLen** 值的大小必须能够显示字段可容纳的最大数字以及小数点 (如果该数字具有小数位的话)。CHAR、DBCHAR、MBCHAR 或 UNICODE 类型的字段的 **fieldLen** 值的大小必须能够处理双字节字符以及任何 shift-in/shift-out 字符。

fieldLen 的缺省值是显示基本类型 (包括所有格式字符) 所需的最大字节数。

对于变量文本字段

下列属性对变量文本字段有用:

cursor = no, cursor = yes

指示第一次显示表单时, 屏幕上的光标是否位于字段的开头。在表单中, 只有一个字段可以将 **cursor** 属性设置为 *yes*。缺省值为 *no*。

modified = no, modified = yes

指示程序是否认为字段已被修改, 而无论用户是否更改了值。有关详细信息, 请参阅 *已修正数据标记* 和 *modified* 属性。

缺省值为 *no*。

protect = no, protect = skip, protect = yes

指定用户是否可以访问该字段。有效值如下所示:

no (变量字段的缺省值)

设置字段, 使用户可以覆盖其中的值。

skip (常量字段的缺省值)

设置字段, 使用户不能覆盖其中的值。另外, 在下列任何一种情况下, 光标将跳过该字段:

- 用户正在依据跳进顺序在上一个字段中输入, 并且按下 **Tab** 键或者在上一个字段填充了内容; 或者
- 用户正在依据跳进顺序在下一个字段中输入, 并且按下 **Shift Tab** 键。

yes

设置字段, 使用户不能覆盖其中的值。

validationOrder = integer

指示字段在验证顺序中的位置。字段的缺省验证顺序就是字段在屏幕上的顺序, 即从左到右, 从上到下。

对于字段数组

在文本和打印表单上, 支持一维数组。在数组声明中, **occurs** 属性的值大于 1, 如下示例所示:

```
myArray char(1)[3];
```

数组元素的定位与您对数组中第一个元素指定的位置有关。缺省行为是将元素垂直地定位于连续的行上。

使用下列属性来更改缺省行为:

columns = numberOfElements

每行中的数组元素数目。缺省值为 1。

linesBetweenRows = numberOfLines

包含数组元素的各行之间的行数。缺省值为 0。

spacesBetweenColumns = numberOfSpaces

每个数组元素之间的空格数。缺省值为 1。

indexOrientation = down, indexOrientation = across

指定程序如何引用数组元素:

- 如果将 **indexOrientation** 设置为 *down*, 则先从上到下然后从左到右对元素进行编号, 从而使给定列中的元素具有顺序编号。缺省情况下, **indexOrientation** 的值为 *down*。
- 如果将 **indexOrientation** 设置为 *across*, 则先从左到右然后从上到下对元素进行编号, 从而使给定行中的元素具有顺序编号。

可以覆盖数组元素的属性。例如, 在以下字段声明中, 在 **myArray** 的第二个元素中覆盖了 **cursor** 属性:

```
myArray char(10)[5]
{position=[4,61], protect=skip, myArray[2] { cursor = yes } };
```

相关概念

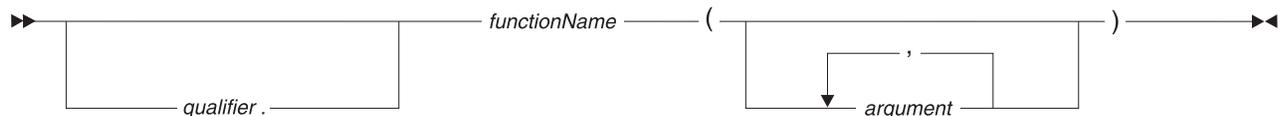
- 第 148 页的『已修正数据标记和 `modified` 属性』
- 第 59 页的『EGL 属性概述』
- 第 144 页的『打印表单』
- 第 20 页的『对部件的引用』
- 第 145 页的『文本表单』
- 第 25 页的『Typedef』

相关参考

- 第 60 页的『字段显示属性』
- 第 61 页的『定义格式的属性』
- 第 612 页的『命名约定』
- 第 47 页的『NUM』
- 第 31 页的『基本类型』
- 第 721 页的『`displayMsgNum()`』
- 第 62 页的『验证属性』

函数调用

函数调用运行 EGL 生成的函数或系统函数。当被调用函数结束时，将继续处理调用之后的语句，或者在复杂情况下，会继续处理表达式或自变量列表中所需的下一进程。



qualifier

下列其中一个符号:

- 函数所在的库的名称; 或者
- 函数所在的包的名称, (可选) 后面跟着一个句点以及函数所在的库的名称。
- *this* (标识当前程序中的函数)

有关限定符并非必需的情况的详细信息, 请参阅[对部件的引用](#)。

function name

被调用函数的名称。

argument

下列其中一项:

- 文字
- 常量
- 变量
- 较为复杂的数字、文本或日期时间表达式, 可能包括函数调用或子串; 但该参数的访问修饰符必须为 `IN`

某些变量作为自变量传递至 EGL 生成的函数, 对于这类变量的影响, 取决于参数是用 `IN`、`OUT` 还是用 `INOUT` 修饰的。有关详细信息, 请参阅[函数参数](#)。

如果被调用函数返回一个值, 可以下列方式使用该调用:

- 作为完整的 EGL 语句（在此情况下函数不返回值并且后跟分号）。
- 作为赋值语句中的源值。
- 作为表达式中的操作数。
- 作为函数调用中的自变量。

在函数调用中调用的函数可能导致变量产生副作用，即在函数甚至在函数调用本身中使用同一变量时，变量值会更改。考虑以下示例：假定函数 Sum 要返回三个自变量的和，并且函数 Increment 以一为增量来增加传递的自变量：

```
b INT = 1;
x INT = Sum( Increment(b), b, Increment(b) );
```

如果 Increment 的自变量与使用 INOUT 修饰的参数有关，则之前语句的结果如下：

- b = 1
- 第一次（最左边）的 Increment 调用会修改值 b，它在从 Increment 返回时等于 2
- Sum 调用中的第二个自变量为 2
- 第二次（最右边）的 Increment 调用会修改值 b，它在从 Increment 返回时等于 3
- 在 Sum 运行后，x 会修改值 7，这是因为该函数中的逻辑使用了值 2、2 和 3

如果 Sum 调用中的第二个自变量与用 INOUT 修饰的参数有关，则该自变量的求值在 Increment 调用之后进行。之前代码的结果如下所示：

- b = 1
- 第一次（最左边）的 Increment 调用会修改值 b，它在从 Increment 返回时等于 2
- 第二次（最右边）的 Increment 调用会修改值 b，它在从 Increment 返回时等于 3
- Sum 中的逻辑开始运行，并且只有此时内存才会与引用的第二个自变量相关联；该内存中的值等于 3
- 在 Sum 运行后，x 会修改值 8，这是因为该函数中的逻辑使用了值 2、3 和 3

一般规则是可通过引用常规表达式求值顺序来标识这些副作用，常规顺序是从左至右，但可以通过使用圆括号来覆盖常规顺序。就像显示的那样，INOUT 的使用更为复杂。

当该参数的访问修饰符为 IN 或 OUT 时，兼容性规则如赋值兼容性中所述。当参数的访问修饰符为 INOUT 时（或者参数在 pageHandler 的 onPageLoad 函数中时），兼容性规则如引用兼容性中所述。

其它规则也适用：

文字

如果访问修饰符为 IN 或 INOUT，可将文字编写为自变量。EGL 生成的代码创建参数类型的临时变量，用值初始化该变量，然后将该变量传递至函数。

固定记录

如果自变量为固定记录，则参数必须为固定记录。

下列规则适用于 basicRecord 类型之外的固定记录：

- 自变量和参数的类型必须完全相同
- 访问修饰符的类型必须为 INOUT

对于类型为 basicRecord 的固定记录，自变量和参数的类型可能会变化：

- 如果访问修饰符的类型为 IN，则自变量的大小必须大于或等于参数的大小。
- 如果访问修饰符的类型为 OUT 或 INOUT，则自变量的大小必须小于或等于参数的大小。

相关概念

- 第 130 页的『函数部件』
- 第 20 页的『对部件的引用』
- 第 690 页的『EGL 语句和命令的语法图』

相关任务

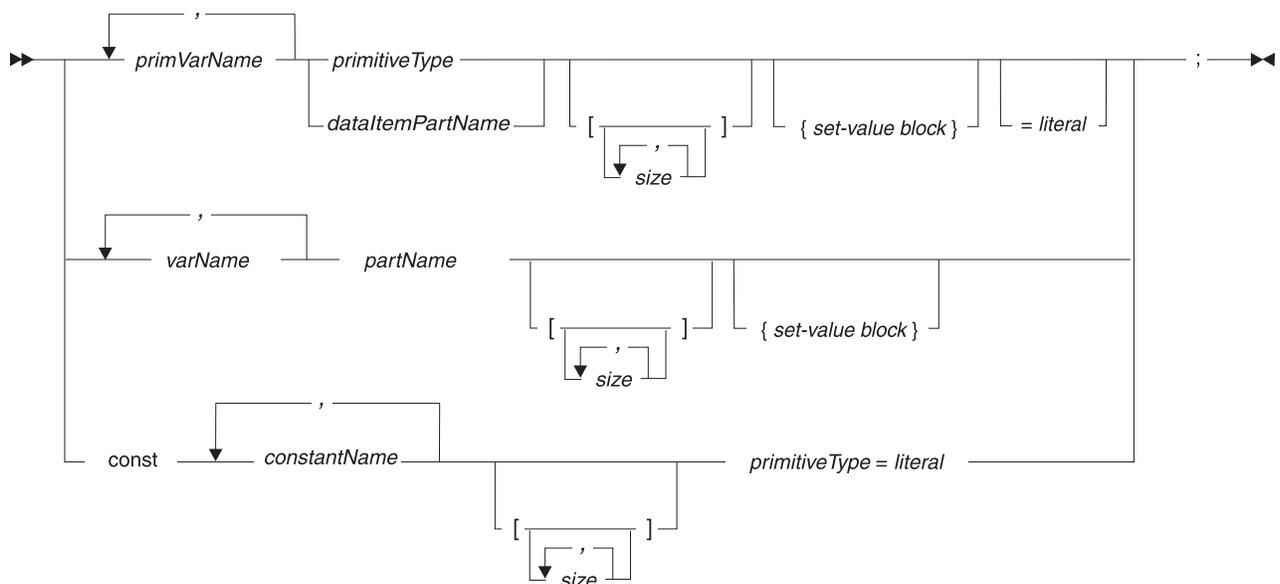
- 第 340 页的『赋值』

相关参考

- 第 335 页的『EGL 中的赋值兼容性』
- 第 80 页的『EGL 语句』
- 第 477 页的『函数参数』
- 第 481 页的『EGL 源格式的函数部件』
- 第 31 页的『基本类型』
- 第 675 页的『EGL 中的引用兼容性』

函数变量

函数中每个变量的语法图如下：



primVarName

指定本地基本变量的名称。有关它们在函数中的用法的详细信息，请参阅对变量和常量的引用。有关其它规则，请参阅命名约定。

primitiveType

基本字段的类型。根据该类型，可能需要下列信息：

- 参数的长度，它是一个整数，表示内存区中的字符或数字的数目。

- 对于某些数字类型，可以指定用来表示小数点后的位数的整数。小数点不与数据存储在一起来。
- 对于类型为 INTERVAL 或 TIMESTAMP 的项，可指定日期时间掩码，它会赋予项值中的给定位置特别的意义（如“年份位”）。

dataItemPartName

对程序可视的 `dataItem` 部件的名称。有关可视性的详细信息，请参阅[对部件的引用](#)。

该部件作为格式模型，如 *Typedef* 所述。

size

数组中的元素数目。如果指定元素数目，则数组是使用该数目的元素进行初始化的。

set-value block

有关详细信息，请参阅 *EGL 属性概述* 和 *Set-value 块*。

= literal

指定基本变量的初始值。

varName

变量名称，可以是基于部件的任何类型。

partName

对程序可视或预定义的部件的名称。有关可视性的详细信息，请参阅[对部件的引用](#)。

该部件作为格式模型，如 *Typedef* 所述。

const constantName primitiveType=literal

常量的名称、类型和值。指定用引号括起来的字符串（对于字符类型）、数字（对于数字类型）或一组相应类型的值（对于数组）。示例如下所示：

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

有关命名规则，请参阅[命名约定](#)。

相关概念

第 130 页的『函数部件』

第 16 页的『部件』

第 20 页的『对部件的引用』

第 53 页的『引用 EGL 中的变量』

第 690 页的『EGL 语句和命令的语法图』

第 25 页的『Typedef』

相关任务

第 481 页的『EGL 源格式的函数部件』

相关参考

第 68 页的『数组』

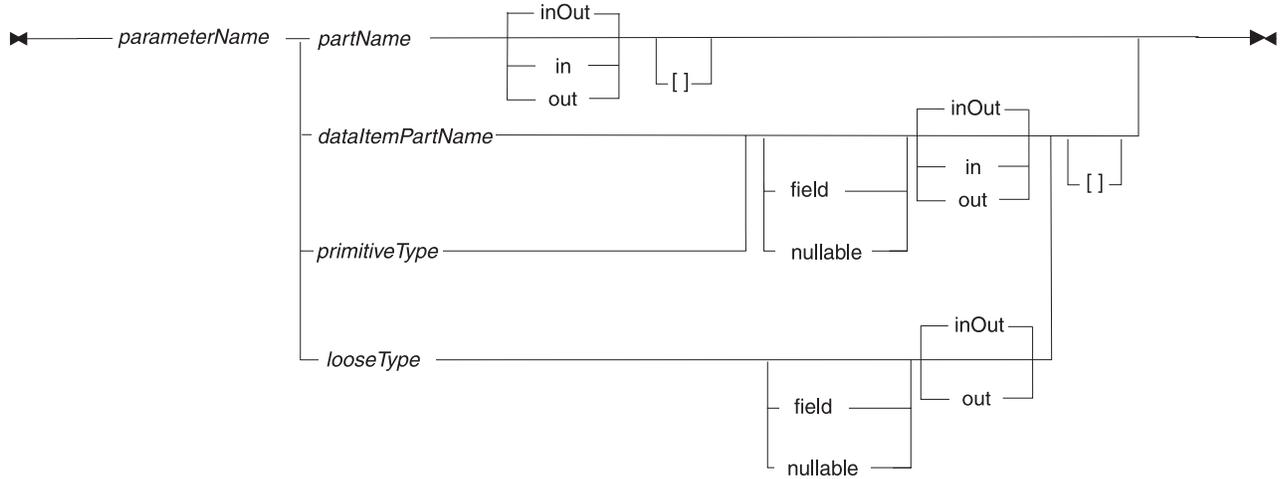
第 38 页的『INTERVAL』

第 612 页的『命名约定』

第 40 页的『TIMESTAMP』

函数参数

函数参数的语法图如下所示:



parameterName

指定参数的名称，该参数可以是记录或数据项；也可以是记录或数据项的数组。有关规则，请参阅命名约定。

如果指定修饰符 **inOut** 或 **out**，对参数值所作的所有更改将体现在调用函数中。这些修饰符稍后将会在 第 479 页的『inOut 及相关修饰符的含义』中描述。

参数对由包含该参数的函数调用的函数不可视；但是，可将参数作为自变量传送到其它函数。

以方括号 ([]) 结尾的参数是动态数组，其它说明声明了该数组每个元素的特征。

inOut

该函数接收自变量值作为输入，而调用程序在该函数结束时接收对该参数所作的所有更改。但是，如果自变量是文字或常量，在处理该自变量时会认为是修饰符 **in** 起作用。

如果参数是某个项并且指定了修饰符 **field**（这指示该参数具有可测试的表单字段属性，如 *blanks* 或 *numeric*），则 **inOut** 修饰符是必需的。

如果参数为记录（不是固定记录），则 **inOut** 修饰符就是唯一有效的修饰符。

如果参数为固定记录，则下列规则适用：

- 如果打算使用该记录访问当前函数（或当前函数调用的函数）中的文件或数据库，则必须指定 **inOut** 修饰符或接受缺省的修饰符。
- 如果记录的类型对于自变量和参数是相同的（例如，如果都是串行记录），则函数中会有特定于记录的状态信息（如文件结束状态）并返回给调用程序，但仅当 **inOut** 修饰符起作用时才会如此

如果 **inOut** 修饰符生效，则相关自变量在引用时必须与该参数相兼容，如 *EGL* 中的引用兼容性中所述。

in 该函数接收自变量值作为输入，而调用程序不受对该参数所作的更改的影响。

不能对已具有修饰符 **field** 的项使用修饰符 **in**。而且不能对记录（不是固定记录）或用于访问当前函数或当前函数调用的函数中的文件或数据库的固定记录指定 **in** 修饰符。

out

该函数不会将自变量值接收为输入；而是根据数据初始化中描述的规则对输入值进行初始化。函数返回时，参数的值将赋给自变量。

如果自变量是文字或常量，在处理该自变量时会认为是修饰符 **in** 在起作用。

不能对具有修饰符 **field** 的参数使用 **out** 修饰符。而且不能对记录或用于访问当前函数或当前函数调用的函数中的文件或数据库的固定记录指定 **out** 修饰符。

partName

对函数可视并且作为参数的 `typeDef`（格式模型）的记录部件。有关哪些部件为可视部件的详细信息，请参阅[对部件的引用](#)。

下列陈述适用于固定记录的输入或输出（I/O）：

- 从同一程序中的另一函数传递的固定记录包括记录状态，如 I/O 错误值 `endOfFile`，但是仅当记录与参数具有相同记录类型时才如此。同样，记录状态的任何更改都将被返回给调用程序，所以，如果对记录参数执行 I/O，则可以在当前函数、调用程序或者由当前函数调用的函数中对该记录进行任何测试。

库函数不接收记录状态。

- 对固定记录执行的任何 I/O 操作都使用对参数指定的记录属性而不是对自变量指定的记录属性。
- 对于 `indexedRecord`、`mqRecord`、`relativeRecord` 或 `serialRecord` 类型的固定记录，与记录声明相关联的文件或消息队列被视为运行单元资源而不是程序资源。每当记录属性 **fileName**（或 **queueName**）具有相同的值时，本地记录声明就共享同一个文件（或队列）。无论在运行单元中有多少个记录与文件或队列相关联，每次都仅有一个物理文件可以与文件或队列名相关联，并且，EGL 在适当情况下可以通过关闭并重新打开文件来确保此规则的实施。

dataItemPartName

对函数可视并且作为参数的 `typeDef`（格式模型）的 `dataItem` 部件。

primitiveType

基本字段的类型。根据该类型，可能需要下列信息：

- 参数的长度，它是一个整数，表示内存区中的字符或数字的数目。
- 对于某些数字类型，可以指定用来表示小数点后的位数的整数。小数点不与数据存储在一起。
- 对于类型为 `INTERVAL` 或 `TIMESTAMP` 的项，可指定日期时间掩码，它会赋予项值中的给定位置特别的意义（如“年份位”）。

looseType

松散类型是一种特殊的基本类型，它仅用于函数参数。如果希望参数接受某个范围的自变量长度，则使用此类型。这样做的好处是可以重复地调用函数并且每次都可以传递长度不同的自变量。

有效值如下所示：

- CHAR
- DBCHAR

- HEX
- MBCHAR
- NUMBER
- UNICODE

如果希望参数接受具有任何基本类型和长度的数字，则指定 **NUMBER** 作为松散类型。在这种情况下，传递给该参数的数字一定不能带有任何小数位。

如果希望参数接受具有特定基本类型但具有任何长度的字符串，则指定 **CHAR**、**DBCHAR**、**MBCHAR**、**HEX** 或 **UNICODE** 作为松散类型并确保该自变量具有相应的基本类型。

自变量的定义确定当函数中的语句对松散类型的参数运行时所发生的情况。

松散类型不可用在库中声明的函数。

有关基本类型的详细信息，请参阅基本类型。

field

指示参数具有表单字段属性，如 *blanks* 或 *numeric*。可以在逻辑表达式中测试那些属性。

仅当指定 **inOut** 修饰符或接受缺省的 **inOut** 修饰符时，**field** 修饰符才可用。

field 修饰符对类型为 `nativeLibrary` 的库中的函数参数不可用。

nullable

指示参数的下列特征：

- 可以将参数设置为 **NULL**
- 该参数能够访问状态信息，该状态信息是对逻辑表达式进行截断或 **NULL** 测试所必需的

仅当传递给参数的自变量是 SQL 记录中的结构项时，**nullable** 修饰符才有意义。下列规则适用：

- 仅当将项属性 **isNullable** 设置为 *yes*
- 无论 **isNullable**
- 不管是修饰符 **inOut**、**in** 还是 **out** 在起作用，都可以指定 **nullable**。

inOut 及相关修饰符的含义

要更好地理解修饰符 **inOut**、**out** 和 **in**，查看以下示例，它会显示不同变量在不同执行位置的值（用注释表示）。

```
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

```

相关概念

- 第 130 页的『函数部件』
- 第 131 页的『类型为 basicLibrary 的库部件』
- 第 131 页的『类型为 basicLibrary 的库部件』
- 第 16 页的『部件』
- 第 20 页的『对部件的引用』
- 第 53 页的『引用 EGL 中的变量』
- 第 25 页的『Typedef』

相关参考

- 第 345 页的『EGL 源格式的基本记录部件』
- 第 430 页的『数据初始化』
- 第 448 页的『EGL 源格式』
- 第 481 页的『EGL 源格式的函数部件』

第 488 页的『EGL 源格式的带索引记录部件』
第 38 页的『INTERVAL』
第 454 页的『逻辑表达式』
第 603 页的『EGL 源格式的 MQ 记录部件』
第 612 页的『命名约定』
第 31 页的『基本类型』
第 675 页的『EGL 中的引用兼容性』
第 676 页的『EGL 源格式的相关记录部件』
第 679 页的『EGL 源格式的串行记录部件』
第 683 页的『EGL 源格式的 SQL 记录部件』
第 40 页的『TIMESTAMP』

EGL 源格式的函数部件

可以在 EGL 文件中声明函数，如 *EGL* 源格式中所述。

以下示例显示了一个带有两个嵌入函数的程序部件，其中还包含一个独立函数和一个独立记录部件：

```
Program myProgram(employeeNum INT)
  {includeReferencedFunctions = yes}

  // program-global variable
  employees record_ws;
  employeeName char(20);

  // a required embedded function
  Function main()

    // initialize employee names
    recd_init();

    // get the correct employee name
    // based on the employeeNum passed
    employeeName = getEmployeeName(employeeNum);
  end

  // another embedded function
  Function recd_init()
    employees.name[1] = "Employee 1";
    employees.name[2] = "Employee 2";
  end

  end

  // standalone function
  Function getEmployeeName(employeeNum INT) returns (CHAR(20))

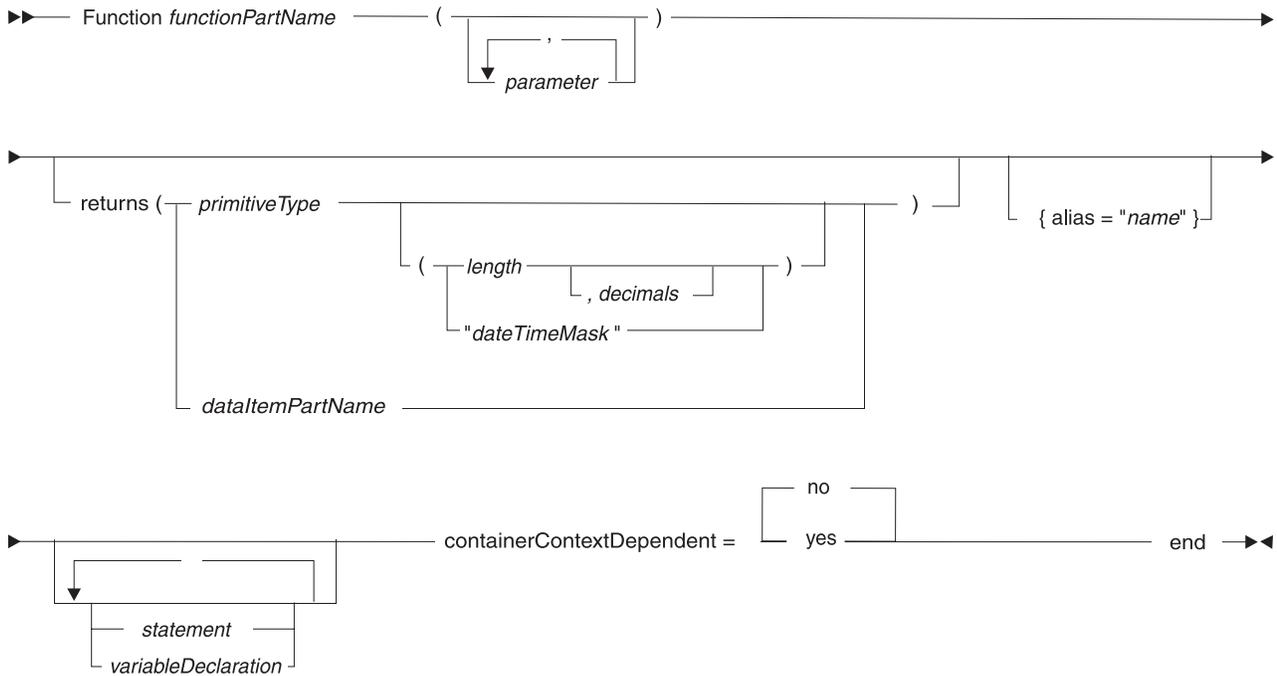
    // local variable
    index BIN(4);
    index = syslib.size(employees.name);
    if (employeeNum > index)
      return("Error");
    else
      return(employees.name[employeeNum]);
    end

  end

end
```

```
// record part that acts as a typeDef for employees
Record record_ws type basicRecord
  10 name CHAR(20)[2];
end
```

函数部件的语法图如下所示:



Function *functionPartName* ... end

将部件标识为函数并指定部件名。有关命名规则，请参阅命名约定。

parameter

一个参数，表示在函数中从头到尾可用的内存区，可从调用函数中接收值。有关用于声明参数的语法的详细信息，请参阅函数参数。

returns (*returnType*)

描述函数返回给调用程序的数据。返回类型的特征必须与调用函数中接收该值的变量的特征相匹配。

{*alias = name*}

仅当函数在类型为 `nativeLibrary` 的库中时才有效。在该上下文中，*name* 是基于 DLL 的函数的名称，并且缺省为 EGL 函数名。如果在用基于 DLL 的函数的名称来命名 EGL 函数时发生了验证错误，则显式设置 **alias** 属性。

dataItemPartName

对函数可视并且作为返回值的 `typeDef`（格式模型）的 `dataItem` 部件。

primitiveType

返回给调用程序的数据的基本类型。

length

返回给调用程序的数据的长度。长度是一个整数，它表示返回的值中的字符或数字的数量。

decimals

对于某些数字类型，可以指定 *decimals*，它是用来表示小数点后的位数的整数。最大小数位数是以下两个数字中较小的那一个：18 或声明为 *length* 的位数。小数点不与数据存储在一起。

"dateTimeMask"

对于 `TIMESTAMP` 和 `INTERVAL` 类型，可指定“*dateTimeMask*”，它会赋予日期时间值中的给定位置特别的意义（如“年份位”）。掩码不会与数据存储在一起。

statement

EGL 语句，如 *EGL* 语句中所述。大多数情况下以分号结尾。

variableDeclaration

变量声明，如函数变量中所述。

containerContextDependent

指示是否扩展用于解析函数的名称空间，这些函数是由正被声明的函数调用的。缺省值为 *no*。

此指示符将在从 VisualAge Generator 迁移的代码中使用。有关详细信息，请参阅 *containerContextDependent*。

相关概念

第 13 页的『EGL 项目、包和文件』

第 130 页的『函数部件』

第 30 页的『Import』

第 131 页的『类型为 basicLibrary 的库部件』

第 131 页的『类型为 basicLibrary 的库部件』

第 16 页的『部件』

第 20 页的『对部件的引用』

第 53 页的『引用 EGL 中的变量』

第 690 页的『EGL 语句和命令的语法图』

第 25 页的『Typedef』

相关参考

第 68 页的『数组』

第 425 页的『containerContextDependent』

第 80 页的『EGL 语句』

第 473 页的『函数调用』

第 477 页的『函数参数』

第 475 页的『函数变量』

第 38 页的『INTERVAL』

第 490 页的『I/O 错误值』

第 612 页的『命名约定』

第 31 页的『基本类型』

第 40 页的『TIMESTAMP』

生成的输出

下表列示生成的输出。有关对每种输出文件指定的名称的详细信息，请参阅生成的输出（参考）。

输出类型	用途	生成类型
构建规划	列示将在目标平台上执行的代码准备步骤	Java 或 Java 包装器
Enterprise JavaBean (EJB) 会话 bean	在 EJB 容器中运行	Java 包装器
Java 程序和相关类	在 J2EE 外部运行, 或者在 J2EE 客户机应用程序、Web 应用程序或 EJB 容器的上下文中运行	Java
Java 包装器	从非 EGL 生成的 Java 代码中调用 EGL 生成的程序	Java 包装器
J2EE 环境文件	提供要插入到 Java 部署描述符中的条目	Java
库 (生成的输出)	提供供其它生成的输出使用的函数和值	Java
链接属性文件	指导如何从生成的 Java 代码进行调用, 但是只有在部署时而不是在生成时此决定才是最终的	Java 或 Java 包装器
PageHandler 部件	创建输出, 该输出控制用户在运行时与 Web 页面进行的交互	Java
程序属性文件	将 Java 运行时属性包含在以下格式中: 仅当在非 J2EE Java 项目中调试 Java 程序时, 才能访问该格式	Java
结果文件	提供有关在目标平台上执行的代码准备步骤的状态信息	Java 或 Java 包装器

相关概念

- 第 1 页的『EGL 简介』
- 第 297 页的『Java 程序、PageHandler 和库』
- 第 315 页的『Java 运行时属性』
- 第 8 页的『运行时配置』

相关任务

- 第 296 页的『构建 EGL 输出』

相关参考

- 『生成的输出 (参考)』

生成的输出 (参考)

EGL 生成的输出很大程度上取决于是生成 Java 还是 Java 包装器。下表显示并非来自特定 EGL 部件的生成的输出的文件名。

输出类型	文件名
第 297 页的『构建规划』	<i>aliasBuildPlan.xml</i>
第 286 页的『EJB 会话 bean』	<i>aliasEJBHome.java</i> (对于 home 接口)、 <i>aliasEJB.java</i> (对于远程 bean 接口) 和 <i>aliasEJBBean.java</i> (对于 bean 实现)
第 324 页的『J2EE 环境文件』	<i>alias-env.txt</i>
第 317 页的『程序属性文件』	<i>alias.properties</i>
第 298 页的『结果文件』	<i>alias_Results_timeStamp.xml</i>

alias

在程序部件中指定的别名（如果有的话）。如果未指定别名，则使用程序部件的名称，但会将该名称截断为运行时环境所允许的最大字符数（如果有必要的话）。

alias 的其它特征由输出类型确定：

- 如果正在生成 Java 程序，则采用 *alias* 中每个字母的大小写，而不更改源代码
- 如果正在生成 Java 包装器，则命名包装器和 EJB 会话 bean 的规则如下所示：
 - *alias* 中的第一个字母是大写的
 - 每个后续字母都是小写的，但以下情况例外：已除去任何下划线或连字符，而且后续字母是大写的

timeStamp

创建文件的日期与时间。该格式反映开发操作系统上的设置。

有关文件名的详细信息，请参阅适当的参考主题：

-
- 第 614 页的『Java 程序生成输出』
- 第 615 页的『Java 包装器生成输出』

相关概念

第 297 页的『构建规划』

第 286 页的『EJB 会话 bean』

第 483 页的『生成的输出』

第 293 页的『生成』

第 324 页的『J2EE 环境文件』

第 317 页的『程序属性文件』

第 298 页的『结果文件』

相关参考

第 614 页的『Java 程序生成输出』

第 615 页的『Java 包装器生成输出』

“生成结果”视图

“生成结果”视图显示代码准备消息，它们是在工作台中执行的生成的结果。这些消息可能是错误、警告或参考消息。仅当从“工作台”中生成时，此视图才是可用的。格式如下所示：

msgid message

msgid

是消息标识。例如，IWN.VAL.4610.e 是 Enterprise Developer 验证错误号 4610 的消息标识。

message

是消息文本。

生成结果将按主部件（程序、PageHandler、表单组、数据表和库）显示在视图中，并且每个部件使用不同的选项卡。结果可以是验证结果与生成结果的组合。

可随时打开此视图，但只有在生成输出后此视图才显示数据。

相关概念

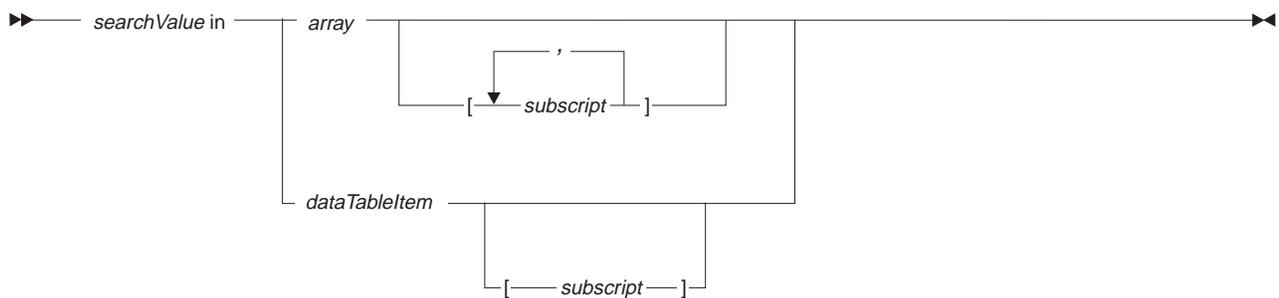
第 7 页的『开发过程』
第 483 页的『生成的输出』
第 293 页的『生成』

相关参考

第 484 页的『生成的输出（参考）』

in 运算符

运算符 **in** 是在具有下列格式的基本逻辑表达式中使用的二目运算符：



searchValue

文字或项，但不是系统变量。

array 一维或多维数组。运算符 **in** 对一维数组进行操作，该数组可以是多维数组的一个元素。

subscript

一个整数，或者是一个解析为整数的项（或系统变量）。下标的值是引用数组中的特定元素的下标（index）。

用作数组下标的项本身不能是数组元素。在下面的每个示例中，`myItemB[1]` 既是下标又是数组元素；因此，下列语法无效：

```
/* the next syntax is not valid */
myItemA[myItemB[1]]

// this next syntax is not valid; but only
// because myItemB is myItemB[1], the
// first element of a one-dimensional array
myItemA[myItemB]
```

dataTableItem

`dataTable` 项的名称。该项表示数据表中的一列。**in** 运算符与该列进行交互，就好像该列是一维数组一样。

如果生成的程序找到了搜索值，则逻辑表达式解析为 `true`。搜索从最后一个数组下标所标识的元素开始。如果 `array` 是一维数组，则最后一个下标是可选的并且缺省设置为 1。如果 `array` 是多维数组，则下列情况成立：

- 必须为每个维提供下标
- 生成的程序搜索由不是最后一个下标的下标序列所标识的一维数组

- 搜索从最后一个下标所标识的元素开始

对于一维数组和多维数组，搜索结束于正在查看的一维数组中的最后一个元素。

在下列任何一种情况下，包含 **in** 的逻辑表达式解析为 **false**：

- 找不到搜索值
- 最后一个下标的值大于正在搜索的一维数组中的条目数

如果基本逻辑表达式解析为 **true**，则 **in** 运算将系统变量 **sysVar.arrayIndex** 设置为包含搜索值的元素的下标值。如果表达式解析为 **false**，则此运算将 **sysVar.arrayIndex** 设置为零。

有关一维数组的示例

假定结构项 **myString** 子结构化为一个有 3 个字符的数组：

```
structureItem name="myString" length=3
  structureItem name="myArray" occurs=3 length=1
```

下表显示了运算符 **in** 的作用（假设 **myString** 为 "ABC"）。

逻辑表达式	表达式的值	sysVar. ArrayIndex 的 值	注释
"A" in myArray	true	1	一维数组的下标缺省为 1
"C" in myArray[2]	true	3	搜索从第二个元素开始
"A" in myArray[2]	false	0	搜索结束于最后一个元素

有关多维数组的示例

假定数组 **myArray01D** 子结构化为一个有 3 个字符的数组：

```
structureItem name="myArray01D" occurs=3 length=3
  structureItem name="myArray02D" occurs=3 length=1
```

在此示例中，**myArray01D** 是一维数组，每个元素都包含一个字符串，该字符串被子结构化为三字符数组。**myArray02D** 是一个二维数组，每个元素（例如 **myArray02D[1,1]**）都包含一个字符。

如果 **myArray01D** 的内容是 "ABC"、"DEF" 和 "GHI"，则 **myArray02D** 的内容如下所示：

```
"A"  "B"  "C"
"D"  "E"  "F"
"G"  "H"  "I"
```

下表显示了运算符 **in** 的作用。

逻辑表达式	表达式的值	sysVar. ArrayIndex 的 值	注释
"DEF" in myArray01D	true	2	引用一维数组不需要下标；缺省情况下，从第一个元素开始搜索

逻辑表达式	表达式的值	sysVar. ArrayIndex 的 值	注释
"C" in myArray02D[1]	—	—	由于引用多维数组必须包括每个维的下标，所以此表达式无效
"I" in myArray02D[3,2]	true	3	搜索从第三行第二个元素开始
"G" in myArray02D[3,2]	false	0	搜索结束于正在查看的行的最后一个元素
"G" in myArray02D[2,4]	false	0	第二个下标大于可供搜索的列数

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 68 页的『数组』

第 454 页的『逻辑表达式』

第 613 页的『运算符和优先顺序』

第 851 页的『arrayIndex』

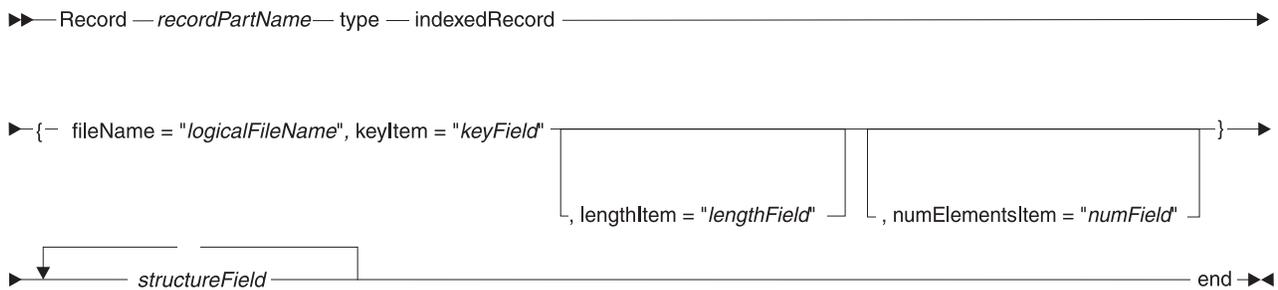
EGL 源格式的带索引记录部件

可以在 EGL 文件中声明类型为 indexedRecord 的记录部件，EGL 源格式对该部件作了描述。

下面是带索引记录部件的示例：

```
Record myIndexedRecordPart type indexedRecord
{
  fileName = "myFile",
  keyItem = "myKeyItem"
}
10 myKeyItem CHAR(2);
10 myContent CHAR(78);
end
```

带索引记录部件的语法图如下所示：



Record recordPartName indexedRecord

将部件标识为具有 indexedRecord 类型并指定名称。有关规则，请参阅命名约定。

fileName = "logicalFileName"

文件名。有关输入的含义的详细信息，请参阅资源关联（概述）。有关规则，请参阅命名约定。

keyItem = "keyItem"

键项，它只能是在同一记录中唯一的结构项。必须对 *keyItem* 使用未限定引用；例如，使用 *myItem* 而不是 *myRecord.myItem*。（然而，在函数中，可以象引用任何结构项一样引用该结构项。）

lengthItem = "lengthItem"

长度项，如支持变长记录的属性中所述。

numElementsItem = "numElementsItem"

元素数目项，如支持变长记录的属性中所述。

structureItem

结构项，如 EGL 源格式的结构项中所述。

相关概念

第 13 页的『EGL 项目、包和文件』

第 20 页的『对部件的引用』

第 16 页的『部件』

第 122 页的『记录部件』

第 53 页的『引用 EGL 中的变量』

第 277 页的『资源关联和文件类型』

第 25 页的『Typedef』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 68 页的『数组』

第 431 页的『EGL 源格式的 DataItem 部件』

第 448 页的『EGL 源格式』

第 481 页的『EGL 源格式的函数部件』

第 603 页的『EGL 源格式的 MQ 记录部件』

第 612 页的『命名约定』

第 31 页的『基本类型』

第 664 页的『EGL 源格式的程序部件』

第 673 页的『支持变长记录的属性』

第 676 页的『EGL 源格式的相关记录部件』

第 679 页的『EGL 源格式的串行记录部件』

第 683 页的『EGL 源格式的 SQL 记录部件』

第 686 页的『EGL 源格式中的结构字段』

I/O 错误值

下表描述影响数据库、文件和 MQSeries 消息队列的输入 / 输出 (I/O) 操作的 EGL 错误值。仅当系统变量 `VGVar.handleHardIOErrors` 设置为 1 时, 与硬错误相关联的值才可供代码使用, 如异常处理中所述。

错误值	错误类型	记录类型	错误值的含义
deadLock	硬错误	SQL	两个程序实例正尝试更改一条记录, 但在系统不介入的情况下这两个程序都无法完成此操作。
duplicate	软错误	带索引记录或相对记录	代码尝试访问具有已存在的键的记录, 并且尝试成功。有关详细信息, 请参阅 <i>duplicate</i> 。
endOfFile	软错误	带索引记录、相对记录或串行记录	有关详细信息, 请参阅 <i>endOfFile</i> 。
ioError	硬错误或软错误	任何	EGL 从 I/O 操作接收到非零返回码。
format	硬错误	任何	访问的文件与记录定义不兼容。有关详细信息, 请参阅 <i>format</i> 。
fileNotAvailable	硬错误	任何	<i>fileNotAvailable</i> 对于任何 I/O 操作都是有可能的, 例如, 它可以指示另一个程序正在使用文件, 或者访问文件所需的资源不足。
fileNotFound	硬错误	带索引记录、消息队列、相对记录或串行记录	找不到文件。
full	硬错误	带索引记录、相对记录或串行记录	在下列情况下将设置 <i>full</i> : <ul style="list-style-type: none">索引文件或串行文件已满
hardIOError	硬错误	任何	发生硬错误, 这是除 <i>endOfFile</i> 、 <i>noRecordFound</i> 或 <i>duplicate</i> 之外的任何错误。
noRecordFound	软错误	任何	有关详细信息, 请参阅 <i>noRecordFound</i> 。
unique	硬错误	带索引记录、相对记录或 SQL 记录	UNQ 指示 <i>unique</i> : 代码尝试添加或替换具有已存在的键的记录, 该尝试失败。有关详细信息, 请参阅 <i>unique</i> 。

duplicate

对于带索引记录或相对记录, 在下列情况下将设置 **duplicate**:

- **add** 语句尝试插入一条记录, 该记录的键或记录标识已存在于文件或备用索引中, 并且插入成功。
- **replace** 语句成功覆盖了记录, 替换值包含与另一记录的备用索引键相同的键。

- **get**、**get next** 或 **get previous** 语句成功读取了记录（或者，表单 *set record position* 的 **set** 语句运行成功），第二条记录具有相同的键。

仅当访问方法返回了信息时才返回 **duplicate** 设置，对于一些操作系统来说，情况如此，而对于其它操作系统，情况并非如此。在 SQL 数据库访问期间，该选项不可用。

endOfFile

在下列情况下将设置 **endOfFile**:

- 当相关文件指针位于文件结尾时，代码对串行记录或相对记录发出 **get next** 语句。当上一个 **get** 或 **get next** 语句访问了文件中的最后一条记录时，指针位于结尾。
- 当相关文件指针位于文件结尾时，代码对带索引记录发出 **get next** 语句，在下列情况下会发生这种情况：
 - 上一个 **get** 或 **get next** 语句访问了文件中的最后一条记录；或者
 - 当发生下列任何一种情况时，上一个类型为 *set record position* 的 **set** 语句访问了文件中的最后一条记录：
 - 键值与文件中最后一条记录的键相匹配；或者
 - 键值中的每个字节都被设置为十六进制 FF。（如果运行类型为 *set record position* 的 **set** 语句时指定了设置为只包含十六进制 FF 的键值，则该语句将位置指针设置到文件结尾。）
- 当相关文件指针位于文件开头时，代码对带索引记录发出 **get previous** 语句，在下列情况下会发生这种情况：
 - 上一个 **get** 或 **get previous** 语句访问了文件中的第一条记录；
 - 代码先前未访问同一个文件；或者
 - 运行类型为 *set record position* 的 **set** 语句时指定了一个键，在文件中，在该键前面没有其它键。
- **get next** 语句尝试从空的或未初始化的文件中检索数据到带索引记录中。
（空文件是其中所有记录均已被删除的文件。未初始化的文件是尚未对其添加任何记录的文件。）
- **get previous** 语句尝试从空文件中检索数据到带索引记录中。

format

任何类型的 I/O 操作都会导致 **format**，设置此错误值的其中一些原因如下所示：

- **记录格式**

文件格式（固定长度或可变长度）与 EGL 记录格式不同。

- **记录长度**

对于定长记录，文件中记录的长度不同于 EGL 记录的长度。对于变长记录，文件中记录的长度大于 EGL 记录的长度。

- **文件类型**

对记录指定的文件类型与运行时的文件类型不匹配。

- **键长度**

文件中的键长度与 EGL 带索引记录中的键长度不同。

- 键偏移

文件中的键位置与 EGL 带索引记录中的键位置不同。

noRecordFound

在下列情况下将设置 `noRecordFound`:

- 对于带索引记录, 找不到与 `get` 语句中指定的键相匹配的记录。
- 对于 EGL 生成的 Java, 当 VSAM 文件是空的或未经初始化时, 代码对带索引记录发出 `get next` 或 `get previous` 语句。
- 对于相对记录, 找不到与 `get` 语句中指定的记录标识相匹配的记录。另一种情况是, `get next` 语句尝试访问位于文件末尾之后的记录。
- 对于 SQL 记录, 找不到与指定的 `SELECT` 语句相匹配的行; 或者当没有剩余可供查看的所选行时遇到了 `get next` 语句。

unique

对于带索引记录或相对记录, 在下列情况下将设置 `unique`:

- `add` 语句尝试插入一条记录, 该记录的键或记录标识已存在于文件或备用索引中, 并且插入由于重复而失败。
- `replace` 语句由于替换值包含与另一记录的备用索引键相同的键而无法覆盖记录。

仅当访问方法返回了信息时才返回 `unique` 设置, 对于一些操作系统来说, 情况如此, 而对于其它操作系统, 情况并非如此。

在 SQL 数据库访问期间, 当正在添加或替换的 SQL 行的键在唯一索引中已存在时, 设置 `unique`。相应的 `sqlcode` 是 -803。

相关参考

第 511 页的『`add`』
第 340 页的『关联元素』
第 517 页的『`close`』

第 520 页的『`delete`』
第 86 页的『异常处理』
第 522 页的『`execute`』
第 533 页的『`get`』
第 544 页的『`get next`』
第 549 页的『`get previous`』
第 454 页的『逻辑表达式』
第 561 页的『`open`』
第 574 页的『`prepare`』
第 576 页的『`replace`』

isa 运算符

运算符 **isa** 是一个二目运算符，用来测试给定表达式是否为特定类型。主要用于测试类型为 ANY 的字段中包含的数据的类型。

该运算符是在具有下列格式的基本逻辑表达式中使用的：

testExpression isa *typeSpecification*

testExpression

数字、文本或日期时间表达式，可能由单个字段或文字组成。

typeSpecification

类型说明，可以是下列任何一项：

- 部件名。
- 基本类型说明（如 STRING）；但是，如果基本类型可能与长度相关联，则必须指定长度，如下列示例中所示：
 - BIN(9)
 - CHAR(5)

不要包括日期时间掩码。

- 后跟成对方括号的类型说明（如上所述）。在此情况下，完整的说明指示特定类型的动态数组、长度（如果适当的话）以及维数。

如果 *testExpression* 与 *typeSpecification* 中标记的类型相匹配，则逻辑表达式解析为 true；否则解析为 false。

相关参考

第 68 页的『数组』

第 454 页的『逻辑表达式』

第 613 页的『运算符和优先顺序』

Java 运行时属性（详细信息）

下表描述了部署描述符或程序属性文件中可以包括的属性，以及生成到 J2EE 环境文件中的值的源（如果有的话）。除非描述列另有声明，否则每个属性的 Java 类型都是 java.lang.String。

运行时属性	描述	生成的值的源
cso.cicsj2c.timeout	<p>指定在使用 CICSJ2C 协议进行调用期间发生超时之前的毫秒数。缺省值为 30000，它表示 30 秒。如果将该值设置为 0，则不会发生超时。该值必须大于等于 0。</p> <p>在此例中，Java 类型是 Java.lang.Integer。</p> <p>当代码正在 WebSphere 390 中运行时，该属性对调用没有任何影响；有关详细信息，请参阅为 CICSJ2C 调用设置 J2EE 服务器。</p>	构建描述符选项 cicsj2cTimeout

运行时属性	描述	生成的值的源
<code>cso.linkageOptions.LO</code>	指定链接属性文件的名称，该文件指导生成程序或包装器如何调用其它程序。 <i>LO</i> 是生成时使用的链接选项部件的名称。有关详细信息，请参阅部署链接属性文件。	<i>LO</i> 来自构建描述符选项 linkage ；缺省值为后跟扩展名 <i>.properties</i> 的链接选项部件的名称
<code>tcpiplistener.port</code>	指定（类 <code>CSOTcpipListener</code> 或 <code>CSOTcpipListenerJ2EE</code> 的）EGL TCP/IP 侦听器用来进行侦听的端口号。不存在缺省值。有关详细信息，请参阅有关设置 <i>TCP/IP</i> 侦听器的主题。 在此例中， <code>Java</code> 类型是 <code>Java.lang.Integer</code> 。	不生成
<code>tcpiplistener.trace.file</code>	指定要在其中记录一个或多个 EGL TCP/IP 侦听器（每个侦听器都具有类 <code>CSOTcpipListener</code> 或 <code>CSOTcpipListenerJ2EE</code> ）的活动的文件名。缺省文件是 <code>tcpiplistener.out</code> 。	不生成；跟踪仅供 IBM 使用
<code>tcpiplistener.trace.flag</code>	指定是否跟踪一个或多个 EGL TCP/IP 侦听器（每个侦听器都具有类 <code>CSOTcpipListener</code> 或 <code>CSOTcpipListenerJ2EE</code> ）的活动。选择下列各项之一： <ul style="list-style-type: none"> • 1 表示将活动记录到在属性 tcpiplistener.trace.flag 中标识的文件中 • 0（缺省值）表示不记录活动 在此例中， <code>Java</code> 类型是 <code>Java.lang.Integer</code> 。有关详细信息，请参阅有关设置 <i>TCP/IP</i> 侦听器的主题。	不生成；跟踪仅供 IBM 使用
<code>vgj.dateMask.gregorian.long.locale</code>	包含在下列两种情况的任何一种情况下使用的日期掩码： <ul style="list-style-type: none"> • 为系统变量 <code>VGVar.currentFormattedGregorianDate</code> 生成的 Java 代码被调用；或者 • EGL 验证长度为 10 或更长的页项或文本表单字段（如果项属性 dateFormat 设置为 <code>systemGregorianCalendarFormat</code> 的话）。 <i>locale</i> 是在属性 vgj.nls.code 中指定的代码。在 Web 应用程序中，可以通过将另一个值赋值给 <code>sysLib.setLocale</code> 来更改所使用的日期掩码属性。	长格里历日期掩码的构建描述符值；缺省值是特定于语言环境的

运行时属性	描述	生成的值的源
<code>vgj.datemask.gregorian.short.locale</code>	<p>包含当 EGL 验证长度小于 10 的页项或文本表单字段（如果项属性 dateFormat 设置为 <i>systemGregorianCalendarFormat</i> 的话）时使用的日期掩码。</p> <p><i>locale</i> 是在属性 vgj.nls.code 中指定的代码。在 Web 应用程序中，可以通过将另一个值赋值给 <code>sysLib.setLocale</code> 来更改所使用的日期掩码属性。</p>	短格里历日期掩码的构建描述符值；缺省值是特定于语言环境的
<code>vgj.datemask.julian.long.locale</code>	<p>包含在下列两种情况的任何一种情况下使用的日期掩码：</p> <ul style="list-style-type: none"> 调用为系统变量 <code>VGVar.currentFormattedJulianDate</code> 生成的 Java 代码；或者 如果项属性 dateFormat 设置为 <i>systemJulianDateFormat</i>，则 EGL 验证页项或文本表单字段的长度是否为 10 或更长。 <p><i>locale</i> 是在属性 vgj.nls.code 中指定的代码。在 Web 应用程序中，可以通过将另一个值赋值给 <code>sysLib.setLocale</code> 来更改所使用的日期掩码属性。</p>	长儒略历日期掩码的构建描述符值；缺省值是特定于语言环境的
<code>vgj.datemask.julian.short.locale</code>	<p>包含当 EGL 验证长度小于 10 的页项或文本表单字段（如果项属性 dateFormat 设置为 <i>systemJulianDateFormat</i> 的话）时使用的日期掩码。</p> <p><i>locale</i> 是在属性 vgj.nls.code 中指定的代码。在 Web 应用程序中，可以通过将另一个值赋值给 <code>sysLib.setLocale</code> 来更改所使用的日期掩码属性。</p>	短儒略历日期掩码的构建描述符值；缺省值是特定于语言环境的
<code>vgj.default.databaseDelimiter</code>	指定用于在系统函数 SysLib.loadTable 和 SysLib.unLoadTable 中隔开每个值的符号。缺省值为竖杠（ ）。	
<code>vgj.default.dateFormat</code>	设置系统变量 StrLib.defaultDateFormat 的初始值；有关有效值的详细信息，请参阅 <i>日期、时间和时间戳记说明符</i>	
<code>vgj.defaultI4GLNativeLibrary</code>	指定类型为 <code>nativeLibrary</code> 的库访问的 DLL 名称。如果未指定库属性 dllName ，则该属性是必需的	
<code>vgj.default.moneyFormat</code>	设置系统变量 StrLib.defaultMoneyFormat 的初始值；有关有效值的详细信息，请参阅 <i>formatNumber()</i>	

运行时属性	描述	生成的值的源
vgj.default.numericFormat	设置系统变量 StrLib.defaultNumericFormat 的初始值；有关有效值的详细信息，请参阅 <i>formatNumber()</i>	
vgj.default.timeFormat	设置系统变量 StrLib.defaultTimeFormat 的初始值；有关有效值的详细信息，请参阅 <i>日期、时间和时间戳记说明符</i>	
vgj.default.timestampFormat	设置系统变量 StrLib.defaultTimestampFormat 的初始值；有关有效值的详细信息，请参阅 <i>日期、时间和时间戳记说明符</i>	
vgj.jdbc.database.SN	<p>指定当通过系统函数 <code>sysLib.connect</code> 或 <code>VGLib.connectionService</code> 来建立数据库连接时使用的 JDBC 数据库名称。</p> <p>该值对于 J2EE 连接和标准（非 J2EE）连接有不同的意义：</p> <ul style="list-style-type: none"> • 对于 J2EE 连接（在生产环境中需要此连接），该值是在 JNDI 注册表中与数据源绑定的名称；例如，<code>jdbc/MyDB</code> • 对于标准 JDBC 连接（可能用于调试），该值是连接 URL；例如，<code>jdbc:db2:MyDB</code> <p>在部署时，当指定 SN 的替换值时，必须定制属性本身的名称。而替换值又必须与包含在 <code>VGLib.connectionService</code> 调用中的服务器名称或包含在 <code>sysLib.connect</code> 调用中的数据库名称相匹配。</p>	要与指定的“服务器名称”相关联的数据库名称的构建描述符值
vgj.jdbc.default.database.autoCommit	指定在每次更改缺省数据库之后是否落实。有效值包括 <code>true</code> 和 <code>false</code> ，如 <i>sqlCommitControl</i> 中所述。	构建描述符选项 sqlCommitControl

运行时属性	描述	生成的值的源
<code>vgj.jdbc.default.database.programName</code>	<p>指定当先前不存在数据库连接时用于 SQL I/O 操作的缺省数据库名称。EGL 将包括程序名（或程序别名，如果有的话）来作为 <i>programName</i> 的替换值，因此，每个程序都有自己的缺省数据库。但是，程序名是可选的，并且名为 <code>vgj.jdbc.default.database</code> 的属性被用作未在此类特定于程序的属性中引用的任何程序的缺省值。</p> <p>对于 J2EE 连接和非 J2EE 连接，此属性本身中的值的含义有所不同：</p> <ul style="list-style-type: none"> • 对于 J2EE 连接，该值是在 JNDI 注册表中与数据源绑定的名称；例如，<code>jdbc/MyDB</code> • 对于标准 JDBC 连接，该值是连接 URL；例如，<code>jdbc:db2:MyDB</code> 	<p>取决于连接类型：</p> <ul style="list-style-type: none"> • 对于 J2EE 连接，构建描述符选项 sqlJNDIName • 对于非 J2EE 连接，构建描述符选项 sqlIDB
<code>vgj.jdbc.default.password</code>	<p>指定用于访问在 vgj.jdbc.default.database 中标识的数据库连接的密码。</p> <p>为了避免在 J2EE 环境文件中泄露密码，执行下列其中一项任务：</p> <ul style="list-style-type: none"> • 使用系统函数 <code>sysLib.connect</code> 或 <code>VGLib.connectionService</code> 来在程序和函数脚本中指定密码；或者 • 在 Web 应用程序服务器中的数据源规范中包括用户标识和密码，如设置 <i>J2EE JDBC</i> 连接中所述。 	构建描述符选项 sqlPassword
<code>vgj.jdbc.default.userid</code>	指定用于访问在 vgj.jdbc.default.database 中标识的数据库连接的用户标识。	构建描述符选项 sqlID
<code>vgj.jdbc.drivers</code>	指定用于访问在 vgj.jdbc.default.database 中标识的数据库连接的驱动程序类。此属性在部署描述符或 J2EE 环境文件中不存在，并且仅用于标准（非 J2EE）JDBC 连接。	构建描述符选项 sqlJDBCdriverClass

运行时属性	描述	生成的值的源
vgj.messages.file	<p>指定包括创建或定制的消息的属性文件。在下列情况下将搜索该文件:</p> <ul style="list-style-type: none"> • 当 EGL 运行时响应函数 <code>SysLib.getMessage</code> 的调用时, 该函数会返回您创建的消息; 有关详细信息, 请参阅 <code>SysLib.getMessage</code> • 当 EGL 运行时处理 consoleUI 应用程序并尝试显示系统变量 ConsoleLib.messageResource 中标识的文件中的帮助或注释文本, 但该变量没有任何值时。 • 当 EGL 尝试显示 Java 运行时消息时, 如 <i>EGL 运行时消息的消息定制</i> 中所述 	
vgj.nls.code	<p>指定程序的三个字母的 NLS 代码。有关有效值的列表, 请参阅 <code>targetNLS</code>。</p> <p>如果未设置此属性, 则下列规则适用:</p> <ul style="list-style-type: none"> • 此值缺省为与缺省 Java 语言环境相对应的 NLS 代码 • 如果缺省 Java 语言环境未与 EGL 支持的任何 NLS 代码相对应, 则值为 ENU 	构建描述符选项 targetNLS
vgj.nls.currency	<p>指定用作货币符号的字符。缺省值由与 vgj.nls.code 相关联的语言环境确定。</p>	构建描述符选项 currencySymbol
vgj.nls.number.decimal	<p>指定用作十进制符号的字符。缺省值由与 vgj.nls.code 相关联的语言环境确定。</p>	构建描述符选项 decimalSymbol

运行时属性	描述	生成的值的源
vgj.properties.file	<p>仅当非 J2EE 运行单元中的第一个程序是使用 VisualAge Generator 或 6.0 之前的 EGL 版本生成的时候，才应使用它。</p> <p>vgj.properties.file 指定另一属性文件。将在非 J2EE 运行单元中全局使用该文件以替代任何非全局程序属性文件。全局文件的使用不受影响。（在使用旧版本 EGL 或 VisualAge Generator 生成第一个程序的运行单元中，全局文件被称为 vgj.properties。）</p> <p>仅当将该属性包括在命令行伪指令中时，才会使用被属性 vgj.properties.file 引用的文件，如下示例所示：</p> <pre>java -Dvgj.properties.file=c:\new.properties</pre> <p>vgj.properties.file 的值包括指向属性文件的标准路径。</p> <p>在属性文件中指定属性 vgj.properties.file 不起作用。</p>	
vgj.ra.QN.conversionTable	<p>指定在访问 <i>QN</i> 标识的 MQSeries 消息队列期间由生成的 Java 程序使用的转换表的名称。有效值为 <code>programControlled</code>、<code>NONE</code> 或转换表名称。缺省值为 <code>NONE</code>。</p>	资源关联属性 conversionTable
vgj.ra.FN.fileType	<p>指定与 <i>FN</i>（这是在记录部件中标识的文件或队列名）相关联的文件的类型。属性值是 <code>seqws</code> 或 <code>mq</code>，如记录和文件类型交叉引用中所述。</p> <p>必须为程序使用的每个逻辑文件指定此部署描述符属性。</p>	资源关联属性 fileType
vgj.ra.FN.replace	<p>指定 <code>add</code> 语句对与 <i>FN</i>（这是记录中标识的文件名）相关联的记录的作用。选择下列两个值中的一个：</p> <ul style="list-style-type: none"> • 1（如果语句替换文件记录的话） • 0（缺省值，如果语句将记录追加至文件的话） <p>在此例中，Java 类型为 <code>java.lang.Integer</code>。</p>	资源关联属性 replace

运行时属性	描述	生成的值的源
vgj.ra.FN.systemName	<p>指定与 <i>FN</i>（这是在记录部件中标识的文件或队列名）相关联的物理文件或消息队列的名称。</p> <p>必须为程序使用的每个逻辑文件指定此部署描述符属性。</p>	资源关联属性 systemName
vgj.ra.FN.text	<p>指定当通过串行记录来访问文件时是否导致生成的 Java 程序执行下列操作：</p> <ul style="list-style-type: none"> 在 add 操作期间追加行结束字符。在非 UNIX 平台上，那些字符是回车符和换行符；在 UNIX 平台上，只能是换行符。 在 get next 操作期间除去行结束字符。 <p><i>FN</i> 是与串行记录相关联的文件名。</p> <p>选择下列其中一个值：</p> <ul style="list-style-type: none"> 1 表示进行更改 0（缺省值）表示禁止进行更改 <p>在此例中，Java 类型为 <code>java.lang.Integer</code>。</p>	资源关联属性 text
vgj.trace.device.option	<p>跟踪数据的目标（如果有的话）。选择下列其中一个值：</p> <ul style="list-style-type: none"> 0 表示写至 <code>System.out</code> 1 表示写至 <code>System.err</code> 2（缺省值）表示写至 vgj.trace.device.spec 中指定的文件，但在以下情况下例外：对于 VSAM I/O 跟踪，写至 <code>vsam.out</code> <p>在此例中，Java 类型为 <code>java.lang.Integer</code>。</p>	生成的值（如果有的话）是 2
vgj.trace.device.spec	<p>指定输出文件的名称（如果 vgj.trace.device.option 设置为 2 的话）。例外情况是：VSAM I/O 跟踪被写至 <code>vsam.out</code>。</p>	生成的值（如果有的话）是 <code>vgjtrace.out</code>

运行时属性	描述	生成的值的源
vgj.trace.type	<p>指定运行时跟踪设置。将感兴趣的值累加起来以获取想要的跟踪:</p> <ul style="list-style-type: none"> • -1 表示全部跟踪 • 0 表示不跟踪 (缺省值) • 1 表示一般跟踪, 包括函数调用和 call 语句 • 2 表示处理数学的系统函数 • 4 表示处理字符串的系统函数 • 16 表示在 call 语句上传递的数据 • 32 表示进行调用时使用的链接选项 • 128 表示 jdbc I/O • 256 表示文件 I/O • 512 表示除 vgj.jdbc.default.password 之外的所有属性 <p>在此例中, Java 类型为 java.lang.Integer。</p>	生成的值 (如果有的话) 是 0

相关概念

第 315 页的『Java 运行时属性』

第 131 页的『类型为 basicLibrary 的库部件』

第 330 页的『链接属性文件』

相关任务

第 329 页的『部署链接属性文件』

第 328 页的『设置 J2EE JDBC 连接』

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

第 326 页的『为 J2EE 应用程序客户机模块中的被调用程序设置 TCP/IP 侦听器』

第 320 页的『为调用的非 J2EE 应用程序设置 TCP/IP 侦听器』

第 240 页的『了解如何建立标准 JDBC 连接』

相关参考

第 370 页的『callLink 元素』

第 351 页的『cicsj2cTimeout』

第 820 页的『connect()』

第 839 页的『connectionService()』

第 863 页的『currentFormattedGregorianCalendar』

第 864 页的『currentFormattedJulianDate』

第 866 页的『currentShortGregorianCalendar』

第 867 页的『currentShortJulianDate』

第 42 页的『日期、时间和时间戳记格式说明符』

第 352 页的『decimalSymbol』

第 801 页的『defaultDateFormat』

第 801 页的『defaultMoneyFormat』

第 802 页的『defaultNumericFormat』

第 802 页的『 defaultTimeFormat 』
第 802 页的『 defaultTimestampFormat 』
第 805 页的『 formatNumber() 』
第 827 页的『 getMessage() 』
第 360 页的『 linkage 』
第 598 页的『 链接属性文件（详细信息） 』
第 828 页的『 loadTable() 』
第 602 页的『 EGL Java 运行时的消息定制 』
第 673 页的『 记录和文件类型交叉引用 』
第 832 页的『 setLocale() 』
第 363 页的『 sqlCommitControl 』
第 364 页的『 sqlDB 』
第 364 页的『 sqlID 』
第 365 页的『 sqlJDBCdriverClass 』
第 365 页的『 sqlJNDIName 』
第 366 页的『 sqlPassword 』
第 368 页的『 targetNLS 』
第 836 页的『 unloadTable() 』

Java 包装器类

当请求将某个程序部件生成成为 Java 包装器时，EGL 会为下列每一项生成包装器类：

- 生成的程序
- 在该程序中被声明为参数的每个固定记录或表单
- 每个被声明为参数的动态数组；并且，如果该数组是固定记录数组，则除了固定记录本身的类以外，还包括动态数组类的类
- 每个具有下列特征的结构项：
 - 位于其中一个为其生成了包装器类的固定记录或表单中
 - 具有至少一个下级结构项；即，具有子结构
 - 是一个数组；在这种情况下，是一个具有子结构的数组

以下是带有具有子结构的数组的固定记录部件的示例：

```
Record myPart type basicRecord
  10 MyTopStructure CHAR(20)[5];
  20 MyStructureItem01 CHAR(10);
  20 MyStructureItem02 CHAR(10);
end
```

在后面的描述中，将给定程序的包装器类称为程序包装器类、参数包装器类、动态数组包装器类和具有子结构的项数组包装器类。

EGL 为每个参数包装器类、动态数组包装器类或具有子结构的项数组包装器类生成一个 BeanInfo 类。BeanInfo 类允许将相关包装器类用作与 Java 相符的 Java bean。您基本上不会与 BeanInfo 类进行交互操作。

在生成包装器时，被调用程序的参数列表不能包括类型为 BLOB、CLOB、STRING、Dictionary、ArrayDictionary 或非固定记录的参数。

关于如何使用包装器类的概述

要使用包装器类来与通过 VisualAge Generator 生成的程序进行通信，在本机 Java 程序中执行下列操作：

- 将类（CSOPowerServer 的子类）实例化以提供中间件服务，例如，在本机 Java 代码与生成的程序之间转换数据：

```
import com.ibm.javart.v6.cso.*;

public class MyNativeClass
{
    /* declare a variable for middleware */
    CSOPowerServer powerServer = null;

    try
    {
        powerServer = new CSOLocalPowerServerProxy();
    }
    catch (CSOException exception)
    {
        System.out.println("Error initializing middleware"
            + exception.getMessage());
        System.exit(8);
    }
}
```

- 将程序包装器类实例化以执行下列操作：
 - 分配数据结构，包括动态数组（如果有的话）
 - 实现对方法的访问，而这些方法又访问生成的程序

对构造函数的调用包括中间件对象：

```
myProgram = new MyprogramWrapper(powerServer);
```

- 声明基于参数包装器类的变量：

```
Mypart myParm = myProgram.getMyParm();
Mypart2 myParm2 = myProgram.getMyParm2();
```

如果程序带有动态数组参数，则声明其它每个都基于动态数组包装器类的变量：

```
myRecArrayVar myParm3 = myProgram.getMyParm3();
```

有关与动态数组进行交互的详细信息，请参阅第 507 页的『动态数组包装器类』。

- 在大多数情况下（如上一个步骤所述），使用参数变量来引用和更改在程序包装器对象中分配的内存
- 设置用户标识和密码，但只有在下列情况下才这样做：
 - Java 包装器通过 iSeries Toolbox for Java 来访问基于 iSeries 的程序；或者
 - 生成的程序在认证远程访问的 CICS for z/OS 区域中运行。

用户标识和密码不用于数据库访问。

通过使用程序对象的 callOptions 变量来设置和查看用户标识和密码，如以下示例所示：

```
myProgram.callOptions.setUserID("myID");
myProgram.callOptions.setPassword("myWord");
myUserID = myProgram.callOptions.getUserID();
myPassword = myProgram.callOptions.getPassword();
```

- 访问生成的程序（在大多数情况下，通过调用程序包装器对象的 `execute` 方法来执行此操作）：

```
myProgram.execute();
```

- 使用中间件对象来建立数据库事务控制，但是仅在以下情况下才执行此操作：
 - 程序包装器对象正在访问 CICS for z/OS 上生成的程序，或者正在通过 IBM Toolbox for Java 来访问基于 iSeries 的 COBOL 程序。在后一种情况下，该调用的 `remoteComType` 值是 JAVA400。
 - 在用来生成包装器类的链接选项部件中，指定了数据库工作单元受客户机（在这种情况下，是包装器）控制；有关详细信息，请参阅“对 `callLink` 元素中的 `luwControl` 的引用”。

如果数据库工作单元受客户机控制，则处理包括使用中间件对象的 `commit` 和 `rollback` 方法：

```
powerServer.commit();
powerServer.rollback();
```

- 关闭中间件对象，并允许进行垃圾回收：

```
if (powerServer != null)
{
    try
    {
        powerServer.close();
        powerServer = null;
    }

    catch(CSOException error)
    {
        System.out.println("Error closing middleware"
            + error.getMessage());
        System.exit(8);
    }
}
```

程序包装器类

程序包装器类包含生成的程序中的每个参数的专用实例变量。如果参数是记录或表单，则变量表示相关参数包装器类的实例。如果参数是数据项，则变量具有基本 Java 类型。

本帮助页面末尾的表描述了 EGL 与 Java 类型之间的转换。

程序包装器对象包含下列公用方法：

- 每个参数的 `get` 和 `set` 方法，其名称的格式如下所示：

```
purposeParmname ()
```

purpose

字 `get` 或 `set`

Parmname

数据项、记录或表单的名称；第一个字母是大写的，其它字母的特征由第 508 页的『Java 包装器类的命名约定』描述的命名约定确定

- 用于调用程序的 `execute` 方法；如果将要作为调用自变量传递的数据位于为程序包装器对象分配的内存中，则使用此方法

可以不对实例变量赋值，而是执行下列操作：

- 为参数包装器对象、动态数组包装器对象以及基本类型分配内存
- 对分配的内存赋值
- 通过调用程序包装器对象的 **call** 方法而不是 **execute** 方法来将这些值传递给程序

程序包装器对象还包含 `callOptions` 变量，该变量具有下列用途：

- 如果您生成 Java 包装器以便在生成时设置调用的链接选项，则 `callOptions` 变量包含链接信息。有关何时设置链接选项的详细信息，请参阅“`callLink` 元素中的 `remoteBind`”。
- 如果您生成 Java 包装器以便在运行时设置调用的链接选项，则 `callOptions` 变量包含链接属性文件的名称。文件名为 `LO.properties`，其中 `LO` 是用于生成的链接选项部件的名称。
- 在以上任何一种情况下，`callOptions` 变量都提供下列方法来设置或获取用户标识和密码：

```

setPassword(password)
setUserid(userid)
getPassword()
getUserid()

```

当将 `callLink` 元素的 **remoteComType** 属性设置为下列其中一个值时，将使用该用户标识和密码：

- CICSECI
- CICSJ2C
- JAVA400

最后，请参照以下情况：当更改了参数或基本类型时，本机 Java 代码需要通知。为了能够发出这样的通知，通过调用程序包装器对象的 **addPropertyChangeListener** 方法来将本机代码注册为侦听器。在这种情况下，下列任何一种情况都会触发 `PropertyChange` 事件，从而导致本机代码在运行时接收到通知：

- 本机代码对具有基本类型的参数调用 **set** 方法
- 生成的程序将已更改的数据返回到具有基本类型的参数中

Sun 公司的 `JavaBean` 规范对 `PropertyChange` 事件作了描述。

参数包装器类集

将为生成的程序中的每个被声明为参数的记录生成参数包装器类。在通常情况下，仅使用参数包装器类来声明引用参数的变量，如以下示例所示：

```
Mypart myRecWrapperObject = myProgram.getMyrecord();
```

在此例中，您正在使用由程序包装器对象分配的内存。

也可以使用参数包装器类来声明内存，如果调用程序对象的 `call` 方法（而不是 `execute` 方法），则必需执行此操作。

参数包装器类包含一组专用实例变量，如下所示：

- 对参数的每个低层结构项包含一个具有 Java 基本类型的变量（但是仅对于既不是数组又不在具有子结构的数组中的结构项才如此）
- 对于每个作为数组并且不具有子结构的 EGL 结构项，包含 Java 基本类型的一个数组

- 本身不在具有子结构的数组中的每个具有子结构的数组的内部数组类的对象；内部类可以具有嵌套的内部类来表示下级具有子结构的数组

参数包装器类包含几个公用方法：

- 一组 **get** 和 **set** 方法允许您获取和设置每个实例变量。每个方法名的格式如下所示：

purpose
siName

purpose

字 **get** 或 **set**。

siName

结构项的名称。第一个字母是大写的，其它字母的特征由第 508 页的『Java 包装器类的命名约定』描述的命名约定确定。

注： 程序调用包含已声明为填充符的结构项；但是，数组包装器类不包含那些结构项的公用 **get** 和 **set** 方法。

- 方法 **equals** 允许您确定存储在同一个类的另一个对象中的值是否与存储在参数包装器对象中的值完全相同。仅当类和值完全相同时，该方法才会返回 **true**。
- 如果当具有 Java 基本类型的变量发生更改时程序需要得到通知，则调用方法 **addPropertyChangeListener**。
- 第二组 **get** 和 **set** 方法允许您获取和设置 SQL 记录参数中的每个结构项的空指示符。这些方法名中的每个方法名的格式如下所示：

purpose
siName
NullIndicator

purpose

字 **get** 或 **set**。

siName

结构项的名称。第一个字母是大写的，其它字母的特征由第 508 页的『Java 包装器类的命名约定』描述的命名约定确定。

具有子结构的项数组包装器类集

具有子结构的项数组包装器类是参数类的内部类，它表示相关参数中具有子结构的数组。具有子结构的项数组包装器类包含一组专用实例变量，这些变量表示数组本身以及该数组下面具有的结构项：

- 对数组的每个低层结构项包含一个具有 Java 基本类型的变量（但是仅对于既不是数组又不在具有子结构的数组中的结构项才如此）
- 对于每个作为数组并且不具有子结构的 EGL 结构项，包含 Java 基本类型的一个数组
- 本身不在具有子结构的数组中的每个具有子结构的数组的内部具有子结构的项数组包装器类的对象；内部类可以具有嵌套的内部类来表示下级具有子结构的数组

具有子结构的项数组包装器类包含下列方法：

- 每个实例变量的一组 **get** 和 **set** 方法

注： 程序调用使用已声明为无名填充符的结构项；但是，具有子结构的项数组包装器类不包含那些结构项的公用 **get** 和 **set** 方法。

- 方法 **equals** 允许您确定存储在同一个类的另一个对象中的值是否与存储在具有子结构的项数组包装器对象中的值完全相同。仅当类和值完全相同时，该方法才会返回 **true**。
- 方法 **addPropertyChangeListener**，如果当具有 Java 基本类型的变量发生更改时程序需要得到通知，则使用此方法

在大多数情况下，参数包装器类中的最顶层具有子结构的项数组包装器类的名称格式为：

```
ParameterClassname.ArrayClassName
```

请参照以下记录，例如：

```
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
```

如果参数 `Company` 基于 `CompanyPart`，则使用字符串 `CompanyPart.Departments` 作为内部类的名称。

内部类的内部类扩展了点线语法的使用。在此示例中，使用符号 `CompanyPart.Departments.Functioncode` 来作为 `Departments` 的内部类名称。

有关如何命名具有子结构的项数组包装器类的其它详细信息，请参阅 *Java 包装器生成的输出*。

动态数组包装器类

将为生成的程序中的每个被声明为参数的动态数组生成动态数组包装器类。请参照以下 EGL 程序特征符：

```
Program myProgram(intParms int[], recParms MyRec[])
```

动态数组包装器类的名称是 `IntParmsArray` 和 `MyRecArray`。

使用动态数组包装器类来声明引用动态数组的变量，如以下示例所示：

```
IntParmsArray myIntArrayVar = myProgram.getIntParms();
MyRecArray myRecArrayVar = myProgram.getRecParms();
```

在为每个动态数组声明变量之后，您可能会添加元素：

```
// adding to an array of Java primitives
// is a one-step process
myIntArrayVar.add(new Integer(5));

// adding to an array of records or forms
// requires multiple steps; in this case,
// begin by allocating a new record object
MyRec myLocalRec = (MyRec)myRecArrayVar.makeNewElement();

// the steps to assign values are not shown
// in this example; but after you assign values,
// add the record to the array
myRecArrayVar.add(myLocalRec);

// next, run the program
```

```

myProgram.execute();

// when the program returns, you can determine
// the number of elements in the array
int myIntArrayVarSize = myIntArrayVar.size();

// get the first element of the integer array
// and cast it to an Integer object
Integer firstIntElement = (Integer)myIntArrayVar.get(0);

// get the second element of the record array
// and cast it to a MyRec object
MyRec secondRecElement = (MyRec)myRecArrayVar.get(1);

```

如该示例所建议的那样，EGL 提供了多个用于处理所声明的变量的方法。

动态数组类的方法	用途
<code>add(int, Object)</code>	在 <code>int</code> 指定的位置插入对象并将当前及后续元素向右移。
<code>add(Object)</code>	将对象追加至动态数组末尾。
<code>addAll(ArrayList)</code>	将 <code>ArrayList</code> 追加至动态数组末尾。
<code>get()</code>	检索包含数组中的所有元素的 <code>ArrayList</code> 对象
<code>get(int)</code>	检索位于由 <code>int</code> 指定的位置中的元素
<code>makeNewElement()</code>	分配具有特定于数组的类型的新元素并检索该元素，而不将该元素添加到动态数组中。
<code>maxSize()</code>	检索一个整数，该整数指示了动态数组中的最大元素数目（但不是实际元素数目）
<code>remove(int)</code>	除去位于由 <code>int</code> 指定的位置中的元素
<code>set(ArrayList)</code>	使用指定的 <code>ArrayList</code> 来替换动态数组
<code>set(int, Object)</code>	使用指定的对象来替换位于由 <code>int</code> 指定的位置中的元素
<code>size()</code>	检索动态数组中的元素数目

在下列情况下例外（还有其它一些例外情况）：

- 如果在 `get` 或 `set` 方法中指定无效的下标
- 如果尝试添加（或设置）与数组中每个元素的类不兼容的类的元素
- 如果在数组达到最大数量而无法支持追加的情况下尝试对动态数组添加元素；并且，如果 `addAll` 方法由于此原因而失败，则该方法不添加任何元素

Java 包装器类的命名约定

EGL 创建符合下列规则的名称：

- 如果名称是全大写的，则将所有字母转换为小写。
- 如果名称是关键字，则在它前面添加下划线
- 如果名称包含连字符或下划线，则除去该字符并将下一个字母转换为大写字母
- 如果名称包含美元符号（\$）、at 符号（@）或磅符（#），则将那些字符中的每一个替换为双下划线（__）并在名称前面添加下划线（_）。
- 如果名称被用作类名，则将第一个字母转换为大写字母。

下列规则适用于动态数组包装器类：

- 在大多数情况下，类名基于作为每个数组元素基础的部件声明（数据项、表单或记录）的名称。例如，如果记录部件名为 `MyRec`，并且数组声明是 `recParms myRec[]`，则相关动态数组包装器类名为 `MyRecArray`。
- 如果数组基于没有相关部件声明的项声明，则动态数组类的名称基于数组名。例如，如果数组声明是 `intParms int[]`，则相关动态数组包装器类名为 `IntParmsArray`。

数据类型交叉引用

下表指示生成的程序中的 EGL 基本类型与生成的包装器中的 Java 数据类型之间的关系。

EGL 基本类型	按字符或数字 所计的长度	按字节计的长 度	小数位数	Java 数据类 型	Java 的最大 精度
CHAR	1-32767	2-32766	不适用	字符串	不适用
DBCHAR	1-16383	1-32767	不适用	字符串	不适用
MBCHAR	1-32767	1-32767	不适用	字符串	不适用
UNICODE	1-16383	2-32766	不适用	字符串	不适用
HEX	2-75534	1-32767	不适用	byte[]	不适用
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

相关概念

第 274 页的『Java 包装器』

第 8 页的『运行时配置』

相关任务

第 273 页的『生成 Java 包装器』

相关参考

第 370 页的『callLink 元素』

第 610 页的『如何为 Java 包装器名取别名』

- 第 598 页的『链接属性文件（详细信息）』
- 第 615 页的『Java 包装器生成输出』
- 第 381 页的『callLink 元素中的 remoteBind』

EGL 中的 JDBC 驱动程序需求

JDBC 驱动程序需求根据数据库管理系统而变化，取决于是用于 EGL 调试时还是运行时：

DB2 UDB

DB2 通用驱动程序与 EGL 兼容，但相关的应用程序驱动程序不兼容；具体地说，应用程序驱动程序不能处理包括 forUpdate 选项的 EGL **open** 或 **get** 语句。

IBM 建议您完全不要使用 Net 驱动程序。

如果在 WebSphere Application Server V6.x 中运行 J2EE 应用程序，则需要 DB2 V8.1.6 或更高版本。如果要在 WebSphere V5.x 测试环境中运行这些应用程序，则需要 DB2 V8.1.3 或更高版本。

Informix

最低可接受 Informix JDBC 驱动程序版本为 2.21.JC6。此驱动程序级别不符合 JDBC 3.0，因此它不支持 EGL **open** 语句中的 hold 选项。符合 Informix JDBC 3.0 的驱动程序现在可用并且应该支持 hold 选项。

Oracle

与 Oracle 10i 一起打包的 JDBC 驱动程序是可以接受的。

下列规则适用于与 EGL 一起使用的任何 JDBC 驱动程序：

- 该驱动程序必须支持 JDBC 2.0 或更高版本
- 下列上下文中必须允许值 java.sql.ResultSet.CONCUR_UPDATABLE:
 - 作为 java.sql.Connection.createStatement(int,int) 的第二个自变量
 - 作为 java.sql.Connection.prepareStatement(String,int,int) 和 java.sql.Connection.prepareCall(String,int,int) 的第三个自变量
- 如果希望在 EGL **open** 语句中支持 hold 选项，驱动程序必须支持 JDBC 3.0，并且下列上下文中必须支持值 java.sql.ResultSet.HOLD_CURSORS_OVER_COMMIT:
 - 作为 java.sql.Connection.createStatement(int,int,int) 的第三个自变量
 - 作为 java.sql.Connection.prepareStatement(String,int,int,int) 和 java.sql.Connection.prepareCall(String,int,int,int) 的第四个自变量

对于任何数据库管理系统，第三方或第四方供应商提供的 JDBC 驱动程序是可以接受的。

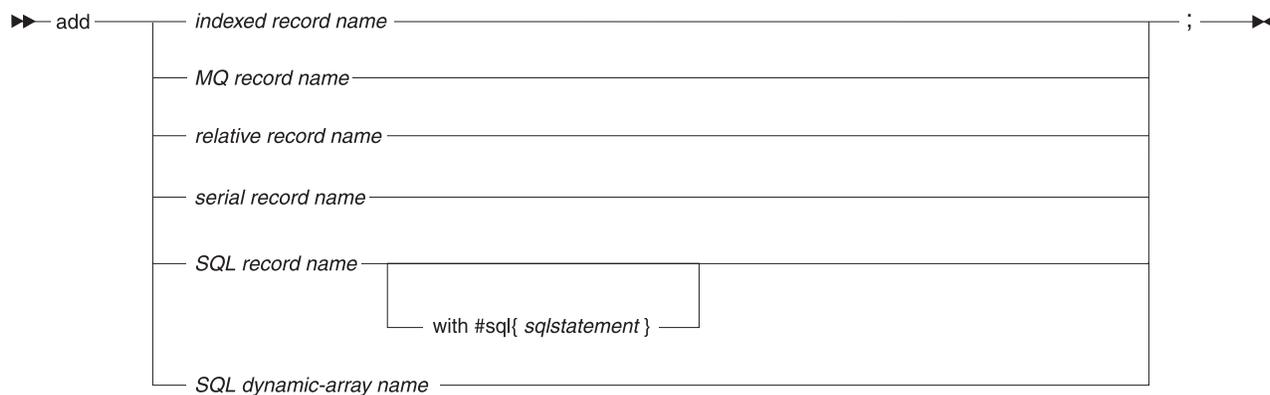
相关任务

- 第 328 页的『设置 J2EE JDBC 连接』
- 第 240 页的『了解如何建立标准 JDBC 连接』

关键字

add

EGL **add** 语句将记录放到文件、消息队列或数据库中；或者将一组记录放到数据库中。



record name

要添加的 I/O 对象的名称：带索引记录、MQ 记录、相对记录、串行记录或 SQL 记录。

with #sql{ sqlStatement }

显式 SQL INSERT 语句。不要在 #sql 之后留下任何空格。

SQL dynamic-array name

SQL 记录动态数组的名称。元素被插入到数据库中，并且每个元素都被插入到由特定于元素的键值指定的位置。在发生第一个错误时，或者在插入了所有元素之后，操作停止。

下面是一个示例：

```
if (userRequest == "A")
  try
    add record1;
  onException
    myErrorHandler(12);
end
end
```

add 语句的行为取决于记录类型。有关 SQL 处理的详细信息，请参阅 *SQL 记录*。

带索引记录

当添加带索引记录时，记录中的键确定了该记录在文件中的逻辑位置。将记录添加至已使用的文件位置将导致硬 I/O 错误 UNIQUE 或者（如果允许重复的话）导致软 I/O 错误 DUPLICATE。

MQ 记录

当添加 MQ 记录时，记录将放置在队列的末尾。进行这种安排的原因是 add 调用了一个或多个 MQSeries 调用：

- MQCONN 将生成的代码与缺省队列管理器连接，并且在无任何活动连接时被调用
- MQOPEN 建立与队列的连接并且在存在活动连接但队列未打开时被调用

- MQPUT 将记录放在队列中，除非在先前的 MQSeries 调用中发生了错误，否则 MQPUT 总是会被调用

相对记录

当添加相对记录时，键项指定文件中的记录的位置。然而，将记录添加至已使用的文件位置会导致硬 I/O 错误 UNIQUE。

记录键项必须可供任何使用该记录的函数使用，并且可以是下列任何一项：

- 同一记录中的项
- 记录中对于程序来说是全局的项，或者对于运行 **add** 语句的函数来说是局部的项
- 对于程序来说是全局的数据项，或者对于运行 **add** 语句的函数来说是局部的数据项

串行记录

当添加串行记录时，记录被放置在文件末尾。

如果生成的程序添加一个串行记录，然后对同一个文件发出 **get next** 语句，则在执行 **get next** 语句之前，程序将关闭并重新打开该文件。因此，跟随在 **add** 语句之后的 **get next** 语句读取文件中的第一条记录。当 **get next** 和 **add** 语句位于不同的程序中，并且其中一个程序调用另一个程序时，也会发生这种行为。

建议避免在多个程序中同时打开同一个文件。

SQL 记录

一些错误状态如下所示：

- 指定了不是 INSERT 类型的 SQL 语句
- 指定了 SQL INSERT 语句的一些子句但不是所有子句
- 指定了具有下列任何特征的 SQL INSERT 语句（或接受隐式 SQL 语句）：
 - 与多个 SQL 表相关
 - 只包含已被声明为只读主变量的主变量
 - 与不存在或者与相关主变量不兼容的列相关联

当在不指定显式 SQL 语句的情况下添加 SQL 记录时，结果如下所示：

- 生成的 SQL INSERT 语句的格式类似于：

```
INSERT INTO tableName
  (column01, ... columnNN)
  values (:recordItem01, ... :recordItemNN)
```

- 记录中的键值确定表中的数据的逻辑位置。没有键的记录是根据 SQL 表定义和数据库的规则来处理的。
- 作为记录部件中的记录项与 SQL 表列相关联的结果，生成的代码将每个记录项中的数据放到相关 SQL 表列中。
- 如果已将记录项声明为是只读的，则生成的 SQL INSERT 语句不包含该记录项，并且数据库管理系统会将相关 SQL 表列的值设置为定义列时所指定的缺省值。

下面是使用 SQL 记录动态数组的示例：

```
try
  add employees;
onException
  sysLib.rollback(); end
```

相关概念

第 20 页的『对部件的引用』
第 125 页的『记录类型和属性』
第 209 页的『SQL 支持』

相关任务

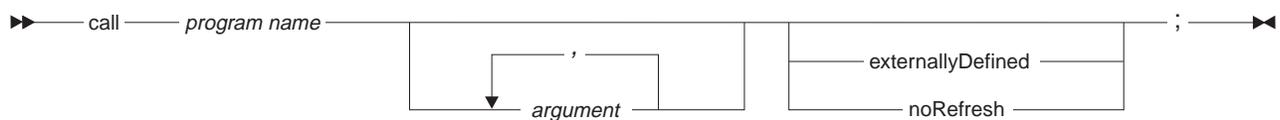
第 690 页的『EGL 语句和命令的语法图』

相关参考

第 517 页的『close』
第 520 页的『delete』
第 533 页的『get』
第 544 页的『get next』
第 549 页的『get previous』
第 86 页的『异常处理』
第 522 页的『execute』
第 490 页的『I/O 错误值』
第 561 页的『open』
第 574 页的『prepare』
第 80 页的『EGL 语句』
第 576 页的『replace』
第 61 页的『SQL 项属性』

call

EGL `call` 语句将控制权转移至另一程序并可以选择传递一系列值。当被调用程序结束时，控制返回到调用程序；并且，如果被调用程序更改了任何通过变量传递的数据，则可用于调用程序的存储区域也会更改。



program name

被调用程序的名称。程序或者是由 EGL 生成的，或者被认为是以外部方式定义的。

指定的名称不能是保留字。如果调用程序必须调用与 EGL 保留字同名的非 EGL 程序，则在 `call` 语句中使用另一个程序名，然后使用链接选项部件的 `callLink` 元素来指定别名，此别名是在运行时使用的名称。

如果被调用程序是 Java 程序，则被调用程序名区分大小写；`calledProgram` 与 `CALLEDPROGRAM` 不同。除此之外，是否区分程序名的大小写取决于被调用程序所在的系统：对于 UNIX 区分大小写，其它则不区分大小写。

在 EGL 调试器中，被调用程序的名称不区分大小写。

argument

一系列值引用之一，用逗号将每个自变量与下一个自变量分开。自变量可以是基本变量、表单、记录、固定记录、非数字文字、非数字常量或（如果 EGL 在

生成时具有对被调用程序的访问权)较复杂的日期时间、数字或文本表达式。不能传递类型为 ANY、ArrayDictionary、Blob、Clob、DataTable 或 Dictionary 的字段。也不能传递这些类型的数组或包括其中任何类型的记录。

externallyDefined

指示程序以外部方式定义的指示符。仅当设置了 VisualAge Generator 兼容性的项目属性时,此指示符才可用。

建议不是在 **call** 语句中而是在生成时使用的链接选项部件中将非 EGL 生成的程序标识为以外部方式定义。(相关属性位于链接选项部件的 callLink 元素中,并且也名为 **externallyDefined**。)

noRefresh

一个指示符,它指示当被调用程序返回控制权时将要避免刷新屏幕。

如果(在开发时)选择了程序属性 **VAGCompatibility** 或者(在生成时)将构建描述符选项 **VAGCompatibility** 设置为 *yes*,则支持此指示符。

如果调用程序位于将文本表单显示到屏幕上的运行单元中,并且存在下列任何一种情况时,此指示符是适合的:

- 被调用程序不显示文本表单;或者
- 调用程序在调用之后写全屏文本表单。

建议不要在 **call** 语句中指示屏幕刷新首选项,而是在生成时使用的链接选项部件中进行指示。(相关属性位于链接选项部件的 callLink 元素中,并且名为 **refreshScreen**。)

下面是一个示例:

```
if (userRequest == "C")
  try
    call programA;
  onException
    myErrorHandler(12);
  end
end
```

call 语句中自变量的个数、类型和顺序必须与被调用程序期望的值的个数、类型和顺序相对应。

强烈建议每个自变量中传递的字节数与期望的字节数相同。仅当对长度不匹配进行的运行时更正导致类型不匹配时,该不匹配才会导致错误:

- 如果被调用 Java 程序接收到的字节数太少,则会在所传递数据的结尾填充空格。
- 如果被调用 Java 程序接收到的字节数太多,则会截断所传递数据的结尾。

例如,在将空格添加至类型为 NUM 的数据项时会发生错误,但将空格添加至类型为 CHAR 的数据项时就不会出错。

下列规则适用于文字和常量:

- 所传递文字或常量的大小必须等于接收参数的大小
- 不能将数字文字或常量作为自变量传递
- 可以将只包含单字节字符的文字或常量传递给 CHAR 或 MBCHAR 类型的参数
- 可以将只包含双字节字符的文字或常量传递给 DBCHAR 类型的参数
- 可以将包含单字节字符与双字节字符组合的文字或常量传递给 MBCHAR 类型的参数

所有目标系统都支持递归调用。

调用受到生成时所使用的链接选项部件（如果有的话）的影响。（通过设置构建描述符选项 **linkage** 来包括链接选项部件。）

有关链接的详细信息，请参阅 [链接选项部件](#)。

相关概念

第 282 页的『链接选项部件』

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 80 页的『EGL 语句』

第 86 页的『异常处理』

第 360 页的『linkage』

第 31 页的『基本类型』

case

EGL **case** 语句标记多组语句的开始，在该位置，最多只运行一组语句。**case** 语句等同于在每个 **case** 子句末尾都有一个 **break** 的 C 或 Java **switch** 语句。



criterion

一个项、常量、表达式、文字或系统变量，包括 `ConverseVar.eventKey` 或 `sysVar.systemType`。

如果指定 *criterion*，则每个后续 **when** 子句（如果有的话）都必须包含一个或多个 *matchExpression* 实例。如果未指定 *criterion*，则每个后续 **when** 子句（如果有的话）都必须包含 *logical expression*。

when

只有在下列情况下才会被调用的子句的开头:

- 指定了 *criterion*, 并且 *when* 子句是第一个要包含与 *criterion* 相等的 *matchExpression* 的子句; 或者
- 未指定 *criterion*, 并且 *when* 子句是第一个要包含等于 *true* 的 *logical expression* 的子句。

如果希望 *when* 子句在被调用时不起任何作用, 则编写不带 EGL 语句的该子句。

case 语句可以具有任何数目的 *when* 子句。

matchExpression

下列其中一个值:

- 数字或字符串表达式
- 要与 *ConverseVar.eventKey* 或 *sysVar.systemType* 作比较的符号

matchExpression 值的基本类型必须与条件值的基本类型相兼容。有关兼容性的详细信息, 请参阅逻辑表达式。

logicalExpression

逻辑表达式。

statement

EGL 语句。

otherwise

在未运行任何 *when* 子句时将被调用的子句的开头。

在运行 *when* 或 *otherwise* 子句中的语句之后, 控制权将传递给紧跟在 **case** 语句末尾后面的 EGL 语句。在任何情况下都不会将控制权传递给下一个 *when* 子句。如果未调用任何 *when* 子句, 并且未使用任何缺省子句, 则还将控制权传递给紧跟在 **case** 语句后面的下一个语句, 而不会出现任何错误状态。

下面是 **case** 语句的一个示例:

```
case (myRecord.requestID)
  when (1)
    myFirstFunction();
  when (2, 3, 4)
    try
      call myProgram;
    onException
      myCallFunction(12);
    end
  otherwise
    myDefaultFunction();
end
```

如果单个子句包含多个 *matchExpression* 实例 (在上一个示例中为 2、3 和 4), 则那些实例的求值是从左至右进行的, 并且一旦找到一个与条件值相对应的 *matchExpression*, 求值就会停止。

相关任务

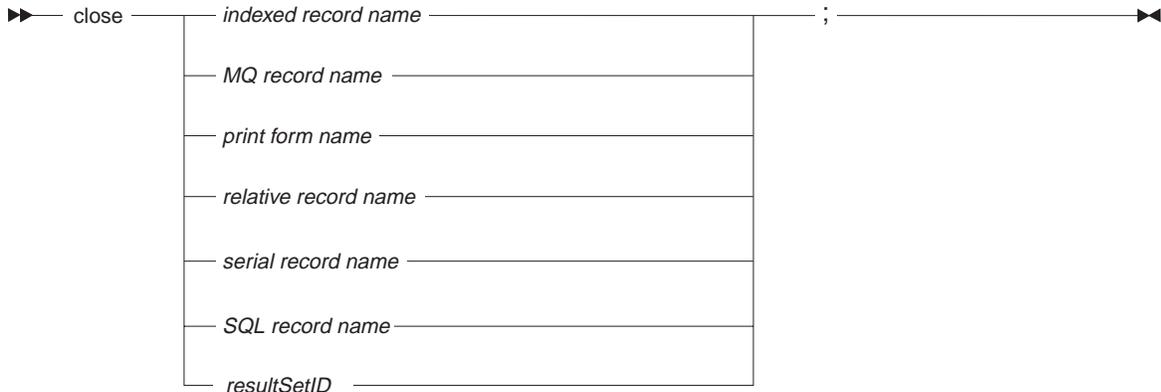
第 690 页的『EGL 语句和命令的语法图』

相关参考

- 第 80 页的『EGL 语句』
- 第 454 页的『逻辑表达式』
- 第 845 页的『eventKey』
- 第 859 页的『systemType』

close

EGL **close** 语句与打印机断开连接；或者关闭与给定记录相关联的文件或消息队列；或者，对于 SQL 记录，关闭由 EGL **open** 或 **get** 语句打开的游标。



name

与在被关闭的资源相关联的 I/O 对象的名称：该对象是打印表单或者带索引记录、MQ 记录、相对记录、串行记录或 SQL 记录

resultSetIdentifier

（仅适用于 SQL 处理）这是一个标识，它使 **close** 语句与同一程序之前运行的 **get** 或 **open** 语句相关。有关详细信息，请参阅 *resultSetID*。

示例：

```

if (userRequest == "C")
  try
    close fileA;
  onException
    myErrorHandler(12);
end
end

```

close 语句的行为取决于 I/O 对象的类型。

带索引记录、串行记录或相对记录

当在 **close** 语句中使用带索引记录、串行记录或相对记录的名称时，EGL 将关闭与该记录相关联的文件。

如果文件已打开并且您使用 *fileAssociation* 项来更改与该文件相关联的资源名称，则 EGL 会在执行下一个影响该文件的语句之前自动关闭该文件。有关详细信息，请参阅 *resourceAssociation*。

当程序结束时，EGL 还会关闭任何已打开的文件。

MQ 记录

当在 **close** 语句中使用 MQ 记录的名称时，EGL 确保会对与该记录相关联的消息队列执行 MQSeries 命令 MQCLOSE。

打印表单

如果 I/O 对象是打印表单，则 **close** 语句发出换页，并且与打印机断开连接或者（如果该打印表单被假脱机至一个文件的话）关闭该文件。

在使用 `ConverseVar.printerAssociation` 来更改打印目标之前，请关闭由 `ConverseVar.printerAssociation` 的当前值指定的打印机或文件。由于可以同时打开多个打印机或打印文件，所以，对每个打印目标发出 **close** 语句选项。

EGL 运行时确保程序结束时所有打印机都已关闭。

SQL 记录

当在 **close** 语句中使用 SQL 记录的名称时，EGL 将关闭对该记录打开的 SQL 游标。

在下列情况下，EGL 自动关闭游标：

- **open** 语句后面跟着一个游标处理循环，该循环一直处理到“找不到记录”（NRF）条件指示已处理集合中的所有行为止
- 当读取了一行并且既未指定 `forUpdate` 也未指定 `singleRow` 来作为选项时，EGL 对 SQL 记录运行 **get** 语句
- EGL 运行 **replace** 或 **delete** 语句，该语句使用由 **get** 语句打开的游标；在这种情况下，在 **get** 语句中指定了 `forUpdate` 作为选项
- EGL 开始对与打开的游标相关联的记录处理 **open** 或 **get** 语句；**close** 优先于其它处理
- 程序运行 `sysLib.commit` 或 `sysLib.rollback`

在下列情况下，EGL 关闭所有已打开的游标：

- 程序具有 `textUI` 类型并且在转换表单之前执行自动落实；有关 `textUI` 程序和 **converse** 语句的详细信息，请参阅分段

相关概念

第 125 页的『记录类型和属性』

第 678 页的『`resultSetID`』

第 147 页的『文本应用程序中的分段』

第 209 页的『SQL 支持』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 511 页的『`add`』

第 520 页的『`delete`』

第 80 页的『EGL 语句』

第 86 页的『异常处理』

第 522 页的『`execute`』

第 533 页的『`get`』

- 第 544 页的『 get next 』
- 第 549 页的『 get previous 』
- 第 490 页的『 I/O 错误值 』
- 第 561 页的『 open 』
- 第 574 页的『 prepare 』
- 第 576 页的『 replace 』
- 第 786 页的『 recordName.resourceAssociation 』
- 第 61 页的『 SQL 项属性 』
- 第 818 页的『 commit() 』
- 第 830 页的『 rollback() 』
- 第 846 页的『 printerAssociation 』
- 第 860 页的『 terminalID 』

continue

EGL **continue** 语句将控制权传送到自身包含 **continue** 语句的 **for**、**forEach** 或 **while** 语句的结束位置。包含语句会继续执行还是结束取决于通常在包含语句开始时引导的逻辑测试。

continue 语句必须与包含语句在同一函数中。



for、forEach 或 while

标识指定类型的最内层包含语句。如果指定其中一种语句类型，则 **continue** 语句必须包含在具有该类型的语句中。如果未指定语句类型，则结果将为如下所示：

- **continue** 语句将控制权传送到最内层 **for**、**forEach** 或 **while** 包含语句的结束位置；并且
- **continue** 语句必须包含在具有其中一种类型的语句中。

相关任务

第 690 页的『 EGL 语句和命令的语法图 』

相关参考

第 80 页的『 EGL 语句 』

converse

EGL **converse** 语句在文本应用程序中显示文本表单。

程序将等待用户响应、接收用户提供的文本表单并继续处理跟随在 **converse** 语句后面的语句。

有关文本表单处理的概述，请按顺序参阅下列各页：

1. 文本表单
2. 分段

►► converse — *textFormName* ————— ; ◀◀

textFormName

对程序可视的文本表单的名称。有关可视性的详细信息，请参阅[对部件的引用](#)。

下面是一个示例：

```
converse myTextForm;
```

存在下列注意事项：

- 对于文本表单，**converse** 语句在被调用程序中始终是有效的；但是，如果正在运行分段的主程序，则 **converse** 语句在下列类型的代码中无效：
 - 具有参数、本地存储器或返回值的函数
 - 由具有参数、本地存储器或返回值的函数调用（直接调用或间接调用）的函数。

相关概念

第 20 页的『[对部件的引用](#)』

第 147 页的『[文本应用程序中的分段](#)』

delete

EGL **delete** 语句从文件中除去一条记录或从数据库中除去一行。

►► delete ————— ; ◀◀

<i>indexed record name</i>
<i>relative record name</i>
<i>SQL record name</i>
from <i>resultSetID</i>

record name

I/O 对象的名称：与正在删除的文件记录或 SQL 行相关联的带索引记录、相对记录或 SQL 记录

from *resultSetID*

一个标识，它使 **delete** 语句与同一程序中之前运行的 **get** 或 **open** 语句相关。有关详细信息，请参阅 *resultSetID*。

下面是一个示例：

```
if (userRequest == "D")
  try
    get myRecord forUpdate;
  onException
```

```

        myErrorHandler(12); // exits the program
    end

    try
        delete myRecord;
        onException
            myErrorHandler(16);
        end
    end

end

```

delete 语句的行为取决于记录类型。有关 SQL 处理的详细信息，请参阅 *SQL 记录*。

带索引记录或相对记录

如果要删除带索引记录或相对记录，则执行下列操作：

- 对该记录发出 **get** 语句并指定 **forUpdate** 选项
- 对同一个文件发出 **delete** 语句，并且，在发出这两个语句之间不进行任何 I/O 操作

在发出 **get** 语句之后，对同一文件执行的下一个 I/O 操作的作用如下所示：

- 如果下一个 I/O 操作是对同一个 EGL 记录执行的 **replace** 语句，则在文件中更改该记录
- 如果下一个 I/O 操作是对同一个 EGL 记录执行的 **delete** 语句，则在文件中对该记录设置删除标记
- 如果下一个 I/O 操作是对同一文件执行的 **get**（带有 **forUpdate** 选项），则后续 **replace** 或 **delete** 对新读取的文件记录有效
- 如果下一个 I/O 操作是对同一个 EGL 记录执行的 **get**（不带 **forUpdate** 选项）或者是对同一个文件执行的 **close**，则释放文件记录而不进行更改

有关 **forUpdate** 选项的详细信息，请参阅 *get*。

SQL 记录

对于 SQL 处理，必须使用 EGL **get** 或 **open** 语句上的 **forUpdate** 选项来检索一行以进行后续删除：

- 可以发出 **get** 语句来检索该行；或者
- 可以发出 **open** 语句来选择一组行，然后调用 **get next** 语句来检索所关心的行。

在任何一种情况下，在生成的代码中，EGL **delete** 语句都是由引用游标中的当前行的 SQL **DELETE** 语句表示的。不能修改该 SQL 语句，该语句是按如下方式定义格式的：

```

DELETE FROM tableName
WHERE CURRENT OF cursor

```

如果您希望编写自己的 SQL **DELETE** 语句，则使用 EGL **execute** 语句。

不能使用单个 EGL **delete** 语句来从多个 SQL 表中除去行。

相关概念

第 125 页的『记录类型和属性』

第 678 页的『resultSetID』

第 678 页的『运行单元』

第 209 页的『SQL 支持』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 511 页的『add』

第 517 页的『close』

第 80 页的『EGL 语句』

第 86 页的『异常处理』

『execute』

第 533 页的『get』

第 544 页的『get next』

第 549 页的『get previous』

第 490 页的『I/O 错误值』

第 574 页的『prepare』

第 561 页的『open』

第 576 页的『replace』

第 61 页的『SQL 项属性』

display

EGL **display** 语句将一个文本表单添加到运行时缓冲区中，但不将数据显示到屏幕上。有关运行时行为的详细信息，请参阅文本表单。

注：如果您正在 VisualAge Generator 兼容性方式下工作，则可以发出具有以下格式的语句：

```
display printForm;
```

printForm

对程序可视的打印表单的名称。

在该情况下，**display** 等同于 **print**。

► display — *textFormName* ————— ; ◄

textFormName

对程序可视的文本表单的名称。有关可视性的详细信息，请参阅[对部件的引用](#)。

相关概念

第 20 页的『对部件的引用』

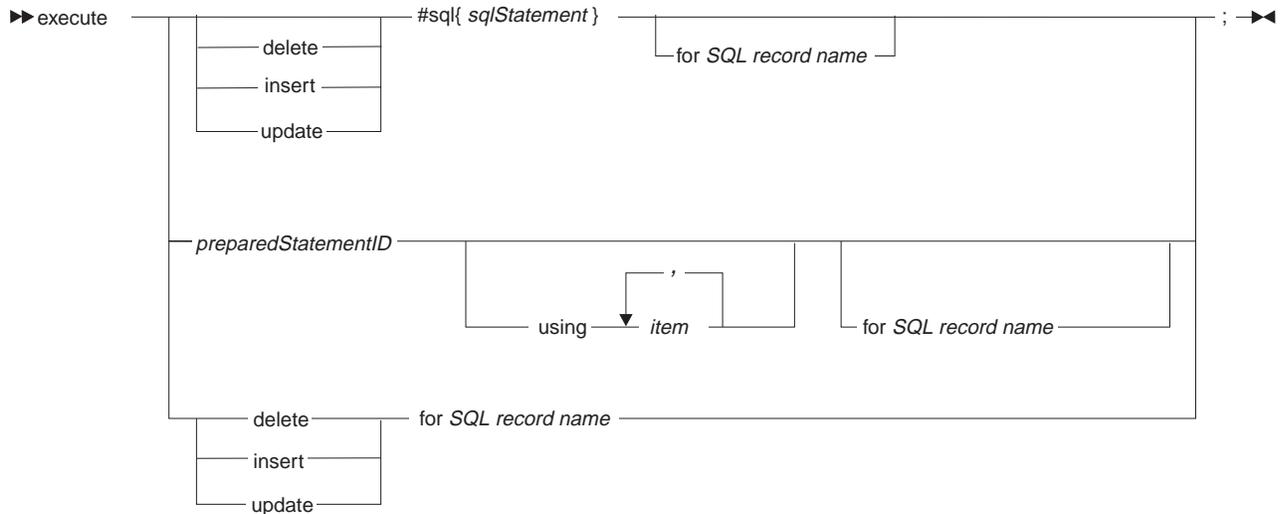
第 145 页的『文本表单』

相关参考

第 576 页的『print』

execute

EGL **execute** 语句允许您编写一个或多个 SQL 语句；特别是，允许您编写 SQL 数据定义语句（例如，CREATE TABLE 类型的数据定义语句）和数据处理语句（例如 INSERT 或 UPDATE 类型的数据处理语句）



#sql{ *sqlStatement* }

显式 SQL 语句。如果要让 SQL 语句更新或删除结果集中的行，则编写包含以下子句的 SQL UPDATE 或 DELETE 语句：

WHERE CURRENT OF *resultSetID*

resultSetID

提供结果集的 EGL open 语句中指定的 *resultSetID*。

不要在 #sql 与左花括号之间留下任何空格。

for *SQL record name*

SQL 记录的名称。

如果指定语句类型 (delete、insert 或 update)，则 EGL 使用 SQL 记录来构建隐式 SQL 语句，如后所述。在任何情况下，都可以使用 SQL 记录来测试操作结果。

preparedStatementID

引用具有指定的标识的 EGL prepare 语句。如果未引用 prepare 语句，则必须指定显式 SQL 语句或者 SQL 记录与语句类型 (delete、insert 或 update) 的组合。

delete, insert, update

指示 EGL 将要提供具有指定类型的隐式 SQL 语句。如果指定语句类型而不是 SQL 记录名，则会发生声明时错误。

如果未设置语句类型，则必须指定显式 SQL 语句或对 prepare 语句的引用。

有关隐式 SQL 语句的概述，请参阅 *SQL 支持*。

下面是几个示例语句（假定 *employeeRecord* 是一个 SQL 记录）：

```
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)
  };
```

可以使用 **execute** 语句来发出下列类型的 SQL 语句:

- ALTER
- CALL
- CREATE ALIAS
- CREATE INDEX
- CREATE SYNONYM
- CREATE TABLE
- CREATE VIEW
- DECLARE global temporary table
- DELETE
- DROP INDEX
- DROP SYNONYM
- DROP TABLE
- DROP VIEW
- GRANT
- INSERT
- LOCK
- RENAME
- REVOKE
- SAVEPOINT
- SET
- SIGNAL
- UPDATE
- VALUES

不能使用 **execute** 语句来发出下列类型的 SQL 语句:

- 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

隐式 SQL DELETE

请求隐式 SQL DELETE 语句的作用是 SQL 记录属性 (**defaultSelectCondition**) 将确定删除哪些表行 (只要每个 SQL 表键列中的值等于 SQL 记录的相应键项中的值)。如果既没有指定记录键也没有指定缺省选择条件, 则删除所有表行。

对特定记录执行的隐式 SQL DELETE 语句类似于以下语句:

```
DELETE FROM tableName
WHERE keyColumn01 = :keyItem01
```

不能使用单个 EGL 语句来从多个数据库表中删除行。

隐式 SQL INSERT

缺省情况下, 请求隐式 SQL INSERT 语句的作用如下所示:

- 记录中的键值确定表中的数据逻辑位置。没有键的记录是根据 SQL 表定义和数据库的规则来处理的。
- 作为记录部件中的记录项与 SQL 表列相关联的结果, 生成的代码将每个记录项中的数据放到相关 SQL 表列中。
- 如果已将记录项声明为是只读的, 则生成的 SQL INSERT 语句不包含该记录项, 并且数据库管理系统会将相关 SQL 表列的值设置为定义列时所指定的缺省值。

隐式 SQL INSERT 语句的格式类似于:

```
INSERT INTO tableName
(column01, ... columnNN)
values (:recordItem01, ... :recordItemNN)
```

一些错误状态如下所示:

- 指定了不是 INSERT 类型的 SQL 语句
- 指定了 SQL INSERT 语句的一些子句但不是所有子句
- 指定了具有下列任何特征的 SQL INSERT 语句 (或接受隐式 SQL 语句):
 - 与多个 SQL 表相关
 - 只包含已被声明为只读主变量的主变量
 - 与不存在或者与相关主变量不兼容的列相关联

隐式 SQL UPDATE

缺省情况下, 请求隐式 SQL UPDATE 语句的作用如下所示:

- 只要每个 SQL 表键列中的值等于 SQL 记录的相应键项中的值, SQL 记录属性 **defaultSelectCondition** 就确定所选择的表行。如果既没有指定记录键也没有指定缺省选择条件, 则更新所有表行。
- 作为 SQL 记录声明中的记录项与 SQL 表列相关联的结果, 给定的 SQL 表列将接收相关记录项的内容。然而, 如果某 SQL 表列与只读的记录项相关联, 则不会更新该列。

对特定记录执行的隐式 SQL UPDATE 语句的格式类似于以下语句:

```
UPDATE tableName
SET   column01 = :recordItem01,
      column02 = :recordItem01, ...
      columnNN = :recordItemNN
WHERE keyColumn01 = :keyItem01
```

在下列任何情况下，将发生错误:

- 所有项都被标识为只读项
- 语句尝试更新多个 SQL 表
- 值正被写至数据库的项与在运行时不存在或与该项不兼容的列相关联

相关概念

第 125 页的『记录类型和属性』

第 209 页的『SQL 支持』

第 20 页的『对部件的引用』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 511 页的『add』

第 517 页的『close』

第 520 页的『delete』

第 80 页的『EGL 语句』

第 86 页的『异常处理』

第 533 页的『get』

第 544 页的『get next』

第 549 页的『get previous』

第 490 页的『I/O 错误值』

第 561 页的『open』

第 574 页的『prepare』

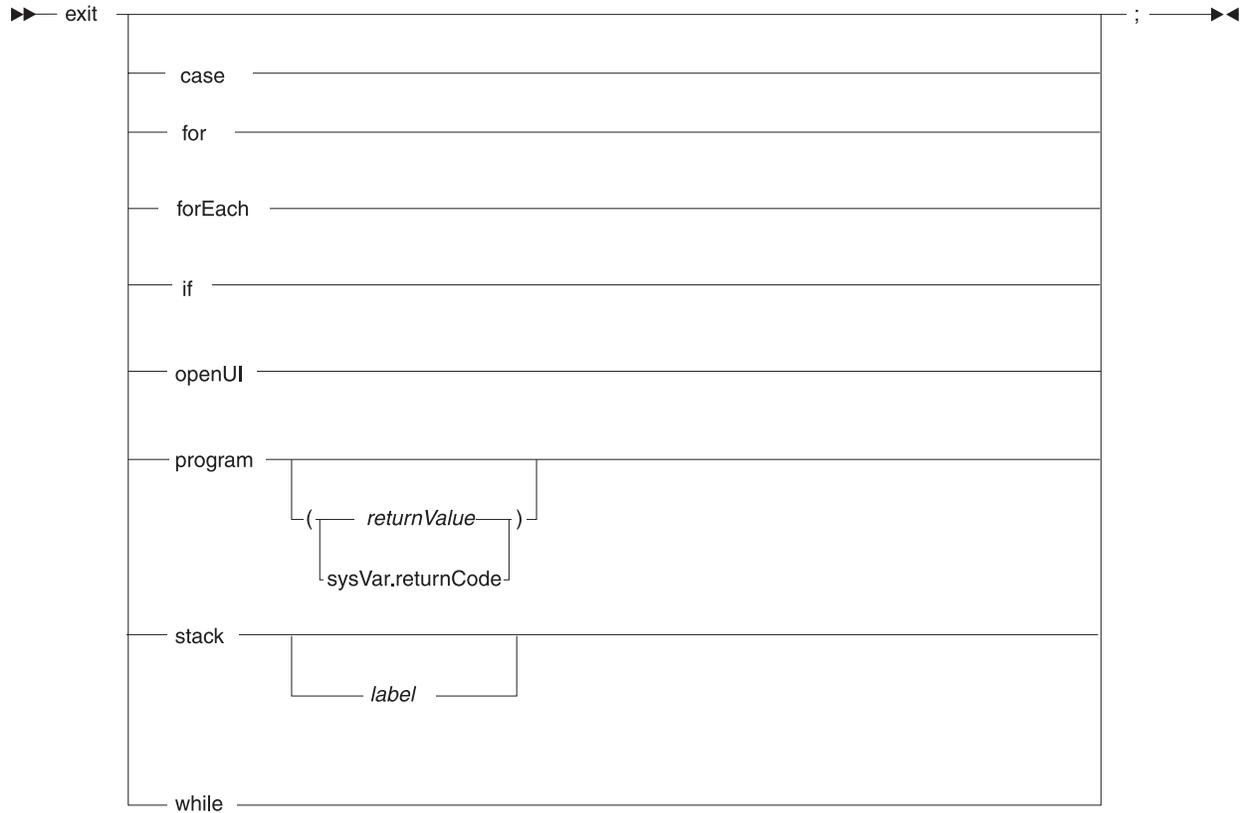
第 576 页的『replace』

第 61 页的『SQL 项属性』

第 860 页的『terminalID』

exit

EGL **exit** 语句离开指定的块，缺省情况下，该块是直接包含 **exit** 语句的块。



case

离开最近进入的 **case** 语句，**exit** 语句位于该语句中。在 **case** 语句之后继续处理。

如果 **exit** 语句并非位于在同一函数中开始的 **case** 语句内，则将发生错误。

for

离开最近进入的 **for** 语句，**exit** 语句位于该语句中。在 **for** 语句之后继续处理。

如果 **exit** 语句并非位于在同一函数中开始的 **for** 语句内，则将发生错误。

forEach

离开最近进入的 **forEach** 语句，**exit** 语句位于该语句中。在 **forEach** 语句之后继续处理。

如果 **exit** 语句并非位于在同一函数中开始的 **forEach** 语句内，则将发生错误。

if

离开最近进入的 **if** 语句，**exit** 语句位于该语句中。在 **if** 语句之后继续处理。

如果 **exit** 语句并非位于在同一函数中开始的 **if** 语句内，则将发生错误。

program

离开程序。

在下列任何情况下，都将把系统变量 **sysVar.returnValue** 中的值返回给操作系统：

- 程序以不包含返回码的 **exit** 语句结束
- 程序以返回 **sysVar.returnValue** 的 **exit** 语句结束
- 程序在不使用终止 **exit** 语句的情况下结束

如果程序以包含除 **sysVar.returnValue** 以外的返回码的终止 `exit` 语句结束，则使用指定的值来代替 **sysVar.returnValue** 可能包含的任何值。

returnValue

文字整数或者解析为整数的项、常量或数字表达式。返回值将提供给操作系统，并且必须在 -2147483648 到 2147483647 之间（包括这两个数字）。

有关返回值的其它详细信息，请参阅 *sysVar.returnValue*。

sysVar.returnValue

一个系统变量，它包含返回给操作系统的值。

有关详细信息，请参阅 *sysVar.returnValue*。

stack

在不设置当前函数的返回值的情况下将控制权返回给主函数。

exit stack 格式的语句将除去对运行时 *stack*（这是函数列表）中的中间函数的所有引用；确切地说，除去对当前函数以及下列函数序列的引用：那些函数的运行使当前函数的运行成为可能。

主函数必须已经调用某个函数（现在在堆栈中），并且调用可能已包括某个参数，而该参数具有修饰符 **out** 或 **inOut**。在这些情况下，格式为 *exit stack* 的 **exit** 语句将参数的值提供给主函数。

如果未指定 *label*（如后面的内容所述），则从主函数中最近运行的函数调用之后的语句继续处理。如果指定了 *label*，则从主函数中标号后的语句继续处理。该 *label* 可以位于主函数中最近运行的函数调用之前或之后。

如果在主函数中指定了 *exit stack* 格式的 `exit` 语句，则处理下一个语句，即使指定了 *label* 亦如此。有关如何转至当前函数中的指定标号的详细信息，请参阅 *goTo*。

label

在主函数中显示的一系列字符，它位于任何块外部，这些块包括：

- `if`
- `else`
- 在 **case** 语句中
- `while`
- `try`

当在继续进行处理的位置显示时，标号的后面有一个冒号。有关标号的有效字符的详细信息，请参阅 *命名约定*。

相关参考

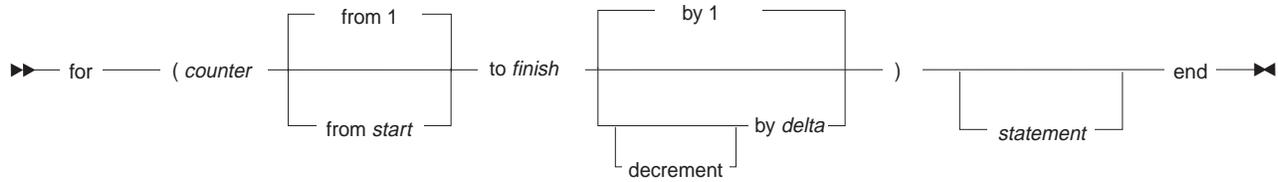
第 554 页的『*goTo*』

第 612 页的『*命名约定*』

第 855 页的『*returnValue*』

for

EGL 关键字 **for** 开始一个语句块，测试求值为 `true` 时该语句块会循环运行多次。循环开始时执行该测试并指示计数器的值是否在指定范围内。关键字 **end** 标记 **for** 语句的结束。



counter

没有小数位的数字变量。**for** 语句中的 EGL 语句可以更改 *counter* 的值。

from *start*

counter 的初始值。如果未指定以 **from** 开头的子句，则初始值为 1。

start 可能是下列任何一项：

- 整数文字
- 没有小数位的数字变量
- 必须解析为整数的数字表达式

to *finish*

如果未指定 **decrement**，则 *finish* 是 *counter* 的上限；如果 *counter* 超出该限制，之前提到的测试将解析为 false，不再执行语句块，并且 **for** 语句结束。

如果指定 **decrement**，则 *finish* 是 *counter* 的下限；如果 *counter* 低于该限制，测试将解析为 false，不再执行语句块，并且 **for** 语句结束。

finish 可能是下列任何一项：

- 整数文字
- 没有小数位的数字变量
- 必须解析为整数的数字表达式

for 语句中的 EGL 语句可以更改 *finish* 的值。

by *delta*

如果未指定 **decrement**，则 *delta* 是在执行 EGL 语句块之后测试 *counter* 的值之前与 *counter* 相加的值。

如果指定 **decrement**，则 *delta* 是在执行 EGL 语句块之后测试 *counter* 的值之前从 *counter* 减去的值。

delta 可能是下列任何一项：

- 整数文字
- 没有小数位的数字变量
- 必须解析为整数的数字表达式

for 语句中的 EGL 语句可以更改 *delta* 的值。

statement

以 EGL 语言表示的语句

下面是一个示例：

```

sum = 0;

// adds 10 values to sum
for (i from 1 to 10 by 1)
  sum = inputArray[i] + sum;
end

```

相关任务

第 690 页的『EGL 语句和命令的语法图』

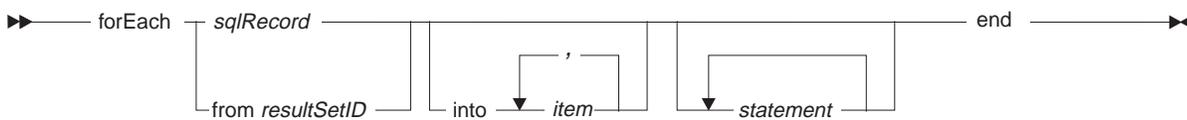
相关参考

第 80 页的『EGL 语句』

forEach

EGL 关键字 **forEach** 标记在循环中运行的一组语句的开始。仅当指定的结果集可用时才会发生第一次迭代。（如果结果集不可用，语句将会因为产生硬错误而失败。）循环读取结果集中的每一行并持续至发生下列其中一个事件：

- 已检索所有行；
- 运行 **exit** 语句；或者
- 发生硬错误或软错误。



sqlRecord

在先前运行的 **open** 语句中使用的 SQL 记录的名称。必须指定 SQL 记录或结果集标识，建议您指定结果集标识。

from *resultSetID*

在先前运行的 **open** 语句中使用的结果集标识。有关详细信息，请参阅 *resultSetID*。

into ... *item*

INTO 子句，它标识从游标或存储过程中接收值的 EGL 主变量。在与此子句类似的子句中（该子句位于 **#sql{ }** 块外部），不要在主变量名称之前包括分号。

此上下文中的 INTO 子句的规范将覆盖相关 **open** 语句中标识的任何 INTO 子句。

statement

以 EGL 语言表示的语句

在大多数情况下，EGL 运行时会发出隐式的 **close** 语句，它会在 **forEach** 语句的最后一次迭代后发生。这一隐式语句会修改 SQL 系统变量，因此您可能想要将与 SQL 有关的系统变量的值保存在 **forEach** 语句的主体中。

如果 **forEach** 语句因为 `noRecordFound` 之外的错误而终止，所以 EGL 运行时不会发出隐式 **close** 语句。

下面是一个示例，更详细的描述可在 *SQL 示例* 中找到：

```

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); // exits program
end

try
forEach (from selectEmp)
empname = empname + " " + "III";

try
execute
#sql{
  update employee
  set empname = :empname
  where current of selectEmp
};
onException
myErrorHandler(10); // exits program
end
end // end forEach; cursor is closed automatically
// when the last row in the result set is read

onException
// the exception block related to forEach is not run if the condition
// is "sqlcode = 100", so avoid the test "if (sqlcode != 100)"
myErrorHandler(8); // exits program
end

sysLib.commit();

```

相关概念

第 678 页的『resultSetID』

第 209 页的『SQL 支持』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 80 页的『EGL 语句』

第 526 页的『exit』

第 561 页的『open』

第 219 页的『SQL 示例』

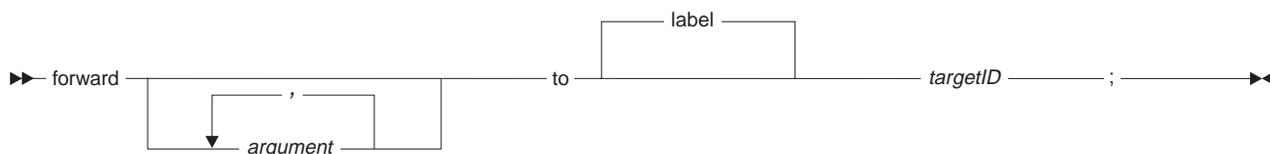
forward

EGL **forward** 语句是从 PageHandler 调用的。主要目的是显示带有变量信息的 Web 页面；但是，该语句也可以调用在 Web 应用程序服务器中运行的 servlet 或 Java 程序。

此语句执行下列操作：

1. 落实可恢复的资源、关闭文件并释放锁
2. 转发控制权
3. 结束运行 **forward** 语句的代码

语法图如下所示:



argument

被传递给正被调用的代码的项或记录。在所有情况下，自变量及其对应参数的名称都必须相同。不能传递文字。

如果正在调用 PageHandler，则自变量必须与对 PageHandler 的 **onPageLoad** 函数指定的参数兼容。函数（如果有的话）可以具有任何有效名称，并可以由 PageHandler 属性 **OnPageLoadFunction** 引用。如果正在调用程序，则自变量必须与程序参数兼容。

根据您的使用技术的方式的不同，您可能会对下列详细信息感兴趣：

- 因为在 Web 应用程序服务器上存储和检索自变量值时将名称用作键，所以自变量必须与对应的参数同名。
- 调用程序在调用 **forward** 语句之前可以执行下列操作，而不是传递自变量：
 - 通过调用系统函数 `J2EELib.setRequestAttr` 来在请求块中放置一个值；或者
 - 通过调用系统函数 `J2EELib.setSessionAttr` 来在会话块中放置一个值。

在这种情况下，接收方不会将该值作为自变量进行接收，而是通过调用适当的系统函数进行接收：

- `J2EELib.getRequestAttr`（访问请求块中的数据）；或者
- `J2EELib.getSessionAttr`（访问会话块中的数据）。
- 字符项是作为类型为 Java 字符串的对象传递的。
- 记录是作为 Java Bean 传递的。

to label *targetID*

指定 Java Server Faces (JSF) 标签，该标签标识基于运行时 JSF 的配置文件中的映射。而映射又标识要调用的对象，该对象是 JSP（通常是与 EGL PageHandler 相关联的 JSP）、EGL 程序、非 EGL 程序或 servlet。字 **label** 是可选的，而 *targetID* 是用引号引起来的字符串。

相关参考

- 第 473 页的『函数调用』
- 第 735 页的『`getRequestAttr()`』
- 第 735 页的『`getSessionAttr()`』
- 第 861 页的『`transferName`』

freeSQL

EGL **freeSQL** 语句释放与动态预编译 SQL 语句相关联的所有资源，关闭与该 SQL 语句相关联的任何打开游标。



preparedStatementID

标识 **prepare** 语句的标识。如果先前未运行该语句，则不会发生错误。

在发出 **freeSQL** 语句后，在没有重新发出 **prepare** 语句的情况下，不能对预编译 SQL 语句运行 **execute**、**open** 或 **get** 语句。

相关概念

第 209 页的『SQL 支持』

相关参考

第 522 页的『execute』

『get』

第 561 页的『open』

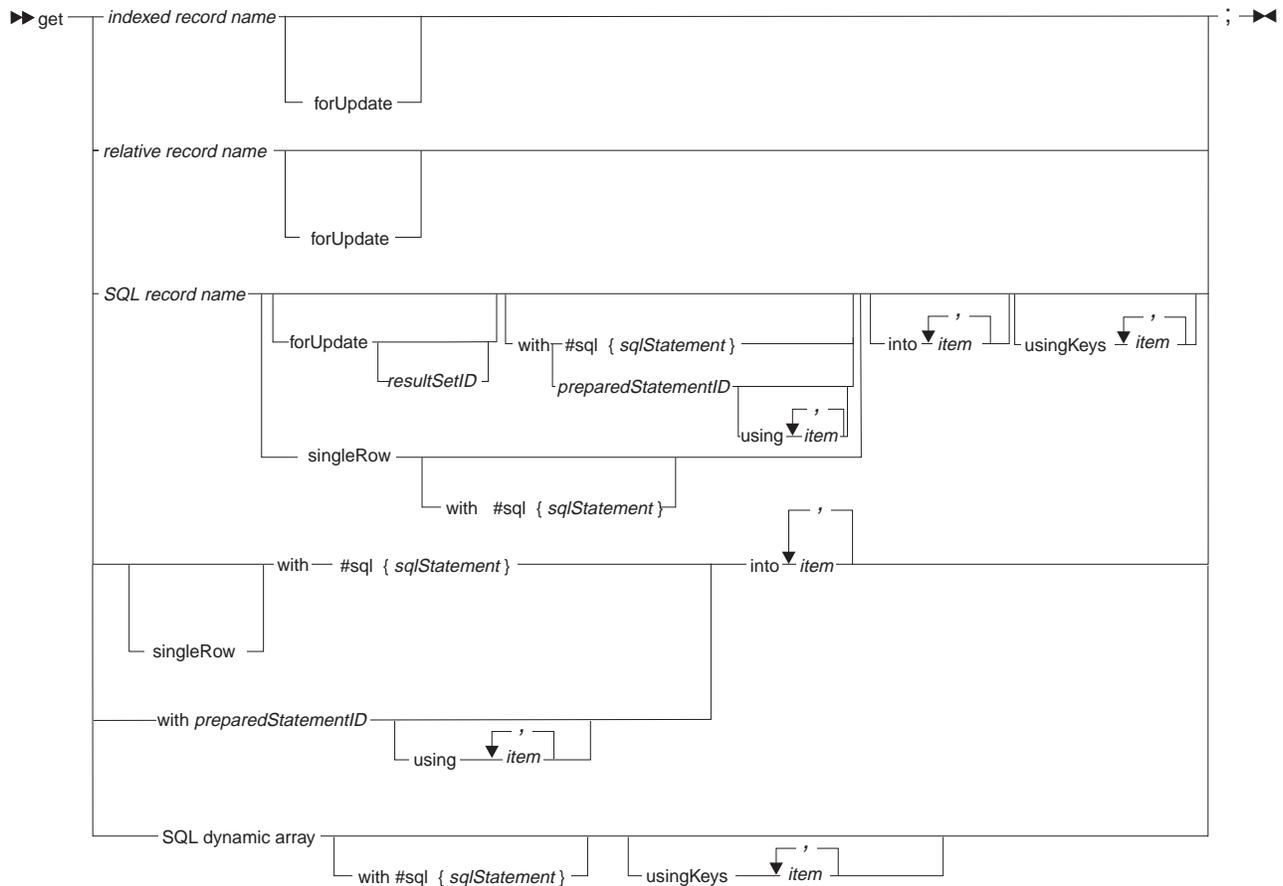
第 574 页的『prepare』

第 690 页的『EGL 语句和命令的语法图』

get

EGL **get** 语句检索单个文件记录或数据库行，并提供了一个选项以允许稍后在代码中替换或删除所存储的数据。另外，此语句允许检索一组数据库行并将每个后续行放到动态数组中的下一个 SQL 记录中。

get 语句有时被标识为 **get by key value**，它与以 *get* 开头的其它语句是不同的。



record name

I/O 对象的名称: 带索引记录、相对记录或 SQL 记录。对于 SQL 处理, 如果未指定 EGL INTO 子句 (稍后描述), 则记录名是必需的。

forUpdate

一个选项, 它允许以后使用 EGL 语句来替换或删除从文件或数据库中检索到的数据。

如果资源是可恢复的 (如对于 VSAM 文件或 SQL 数据库的情况), 则 **forUpdate** 选项锁定记录, 以使它在落实发生之前不能被其它程序更改。有关落实处理的详细信息, 请参阅逻辑工作单元。

resultSetID

结果集标识, 用于 EGL **replace**、**delete** 或 **execute** 语句以及 EGL **close** 语句。有关详细信息, 请参阅 *resultSetID*。

singleRow

一个选项, 它导致生成更有效率的 SQL, 当您确定 **get** 语句中的搜索条件只适用于一行并且不打算更新或删除该行时, 此选项适用。如果在搜索条件适用于多行时指定此选项, 则会导致运行时 I/O 错误。有关其它详细信息, 请参阅 SQL 记录。

#sql{ sqlStatement }

显式的 SQL SELECT 语句, 如 SQL 支持中所述。不要在 #sql 与左花括号之间留下任何空格。

into ... item

EGL INTO 子句, 它标识从关系数据库中接收值的 EGL 主变量。在下列任何一种情况下, 当处理 SQL 时, 此子句是必需的:

- 未指定 SQL 记录; 或者
- 同时指定了 SQL 记录和显式 SQL SELECT 语句, 但 SQL SELECT 子句中的列与记录项无关。(关联位于 SQL 记录部件中, 如 SQL 项属性中所述。)

在与此子句类似的子句中 (该子句位于 **#sql{ }** 块外部), 不要在主变量名称之前包括分号。

preparedStatementID

在运行时准备 SQL SELECT 语句的 EGL **prepare** 语句的标识。 **get** 语句动态地运行 SQL SELECT 语句。有关详细信息, 请参阅 *prepare*。

using ... item

USING 子句, 它标识在运行时可供已准备的 SQL SELECT 语句使用的 EGL 主变量。在类似于此子句的子句 (该子句位于 **sql** 与 **end** 块的外部) 中, 不要在主变量名称之前包括分号。

usingKeys ... item

标识用来构建隐式 SQL 语句中的 WHERE 子句的键值部分的键项列表。如果未指定显式 SQL 语句, 则在运行时将使用隐式 SQL 语句。

如果未指定 **usingKeys** 子句, 则隐式语句的键值部分基于 **get** 语句所引用的 SQL 记录部件, 或者基于作为 **get** 语句所引用的动态数组基础的 SQL 记录部件。

对于动态数组, **usingKeys** 子句中的项 (或者是 SQL 记录中的主变量) 一定不能位于作为动态数组基础的 SQL 记录中。

如果指定显式 SQL 语句, 则将忽略 **usingKeys** 信息。

SQL dynamic array

由 SQL 记录组成的动态数组的名称。

以下示例显示如何读取和替换文件记录:

```
emp.empnum = 1;           // sets the key in record emp

try
  get emp forUpdate;
onException
  myErrorHandler(8); // exits the program
end

emp.empname = emp.empname + " Smith";

try
  replace emp;
onException
  myErrorHandler(12);
end
```

当检索数据库行时, 下一个 **get** 语句将使用 SQL 记录 **emp**, 并且不可能进行后续更新或删除:

```
try
  get emp singleRow into empname with
  #sql{
    select empname
    from Employee
    where empnum = :empnum
  };
onException
  myErrorHandler(8);
end
```

下一个示例使用同一个 SQL 记录来替换 SQL 行:

```
try
  get emp forUpdate into empname with
  #sql{
    select empname
    from Employee
    where empnum = :empnum
  };

onException
  myErrorHandler(8); // exits the program
end

emp.empname = emp.empname + " Smith";

try
  replace emp;
onException
  myErrorHandler(12);
end
```

有关 **get** 语句的详细信息取决于记录类型。有关 SQL 处理的详细信息, 请参阅 *SQL 记录*。

带索引记录

当对带索引记录发出 **get** 语句时, 记录中的键值确定从文件中检索哪个记录。

如果要替换或删除带索引记录（或相对记录），则必须对该记录发出 **get** 语句，然后发出文件更改语句（**replace** 或 **delete**），在此之前不对同一个文件执行任何 I/O 操作。在发出 **get** 语句之后，对同一文件执行的下一个 I/O 操作的效果如下所示：

- 如果下一个 I/O 操作是对同一个 EGL 记录执行的 **replace** 语句，则在文件中更改该记录
- 如果下一个 I/O 操作是对同一个 EGL 记录执行的 **delete** 语句，则在文件中对该记录设置删除标记
- 如果下一个 I/O 操作是对同一文件中的记录执行的 **get** 语句，并包含 **forUpdate** 选项，则后续 **replace** 或 **delete** 语句对新读取的文件记录有效
- 如果下一个 I/O 操作是对同一个 EGL 记录执行的 **get** 语句（不带 **forUpdate** 选项）或者是对同一个文件执行的 **close** 语句，则释放文件记录而不进行更改

如果文件是 VSAM 文件，则 EGL **get** 语句（带有 **forUpdate** 选项）将防止记录被其它程序更改。

相对记录

当对相对记录发出 **get** 语句时，与记录相关联的键项确定从文件中检索哪个记录。该键项必须可供任何使用该记录的函数使用，并且可以是下列任何一项：

- 同一记录中的项
- 记录中对于程序来说是全局的项，或者对于运行 **get** 语句的函数来说是局部的项
- 对于程序来说是全局的数据项，或者对于运行 **get** 语句的函数来说是局部的数据项

如果要替换或删除带索引记录（或相对记录），则必须对该记录发出 **get** 语句，然后发出文件更改语句（**replace** 或 **delete**），在此之前不对同一个文件执行任何 I/O 操作。在发出 **get** 语句之后，对同一文件执行的下一个 I/O 操作的效果如下所示：

- 如果下一个 I/O 操作是对同一个 EGL 记录执行的 **replace** 语句，则在文件中更改该记录
- 如果下一个 I/O 操作是对同一个 EGL 记录执行的 **delete** 语句，则在文件中对该记录设置删除标记
- 如果下一个 I/O 操作是对同一文件执行的 **get**（带有 **forUpdate** 选项），则后续 **replace** 或 **delete** 对新读取的文件记录有效
- 如果下一个 I/O 操作是对同一个 EGL 记录执行的 **get**（不带 **forUpdate** 选项）或者是对同一个文件执行的 **close**，则释放文件记录而不进行更改

SQL 记录

EGL **get** 语句导致生成的代码包含 SQL **SELECT** 语句。如果指定 **singleRow** 选项，则 SQL **SELECT** 语句是独立语句。另外，SQL **SELECT** 语句是游标中的子句，如 *SQL* 支持中所述。

错误状态： 当使用 **get** 语句来从关系数据库中读取数据时，无效的情况包括：

- 指定了不是 **SELECT** 类型的 SQL 语句
- 直接在 SQL **SELECT** 语句中指定 SQL **INTO** 子句
- 除 SQL **INTO** 子句之外，指定了 **SELECT** 语句的一部分但非全部子句
- 指定（或接受）的 SQL **SELECT** 语句与不存在的列相关联，或者和相关主变量不兼容

下列错误状态都是使用 `forUpdate` 选项时可能发生的错误状态:

- 指定 (或接受) 的 SQL 语句意图更新多表; 或者
- 使用了 SQL 记录作为 I/O 对象, 并且所有记录项都是只读的。

并且, 下列情况将导致错误:

- 定制了带有 `forUpdate` 选项的 EGL **get** 语句, 但未指示特定 SQL 表列可用于更新; 并且
- 与该 **get** 语句相关的 `replace` 语句尝试修改该列。

可以通过下列任何一种方法来解决前述不匹配问题:

- 定制 EGL **get** 语句时, 在 SQL `SELECT` 语句的 `FOR UPDATE OF` 子句中包括列名; 或者
- 在定制 EGL `replace` 语句时, 消除对 SQL `UPDATE` 语句的 `SET` 子句中的列的引用; 或
- 对 **get** 和 **replace** 语句接受缺省值。

隐式 SQL SELECT 语句: 当指定 SQL 记录来作为 **get** 语句的 I/O 对象, 但未指定显式 SQL 语句时, 隐式 SQL `SELECT` 具有下列特征:

- 只要每个 SQL 表键列中的值等于 SQL 记录的相应键项中的值, 名为 **defaultSelectCondition** 的特定于记录的属性就确定所选择的表行。如果既没有指定记录键也没有指定缺省选择条件, 则选择所有表行。如果由于任何原因而选择了多个表行, 则将检索到的第一行放在记录中。
- 作为记录项与记录定义中的 SQL 表列相关联的结果, 给定的项接收相关 SQL 表列的内容。
- 如果指定 `forUpdate` 选项, 则 SQL `SELECT FOR UPDATE` 语句不包括只读记录项。
- 除了仅当 **get** 语句包含 `forUpdate` 选项时才存在 `FOR UPDATE OF` 子句以外, 特定记录的 SQL `SELECT` 语句与以下语句类似:

```
SELECT column01,  
       column02, ...  
       columnNN  
FROM   tableName  
WHERE  keyColumn01 = :keyItem01  
FOR UPDATE OF  
       column01,  
       column02, ...  
       columnNN
```

独立 SQL `SELECT` 上的 SQL `INTO` 子句或者与游标相关的 `FETCH` 语句上的 SQL `INTO` 子句与以下子句相似:

```
INTO   :recordItem01,  
       :recordItem02, ...  
       :recordItemNN
```

当未指定 `INTO` 子句时, 如果将 SQL 记录与显式 SQL `SELECT` 语句配合使用, 则 EGL 派生 SQL `INTO` 子句。派生的 `INTO` 子句中的项就是与 SQL 语句的 `SELECT` 子句中列示的列相关联的那些项。(项与列的关联位于 SQL 记录部件中, 如 SQL 项属性中所述。) 如果列未与项相关联, 则 EGL `INTO` 子句是必需的。

当指定 SQL 记录动态数组来作为 **get** 语句的 I/O 对象, 但未指定显式 SQL 语句时, 隐式 SQL `SELECT` 类似于对单个 SQL 记录描述的 SQL `SELECT`, 但有下列区别:

- 查询的键值部分是基于大于等于条件的关系集合:

```
keyColumn01 >= :keyItem01 &  
keyColumn02 >= :keyItem02 &  
.  
.  
.  
keyColumnN >= :keyItemN
```

- **usingKeys** 子句中的项（或者是 SQL 记录中的主变量）一定不能位于作为动态数组基础的 SQL 记录中。

相关概念

第 279 页的『逻辑工作单元』
第 125 页的『记录类型和属性』
第 20 页的『对部件的引用』
第 678 页的『resultSetID』
第 209 页的『SQL 支持』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

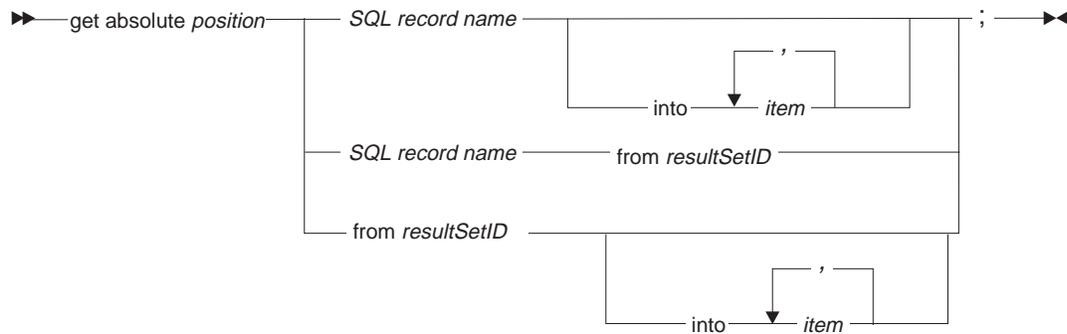
第 511 页的『add』
第 517 页的『close』

第 520 页的『delete』
第 80 页的『EGL 语句』
第 86 页的『异常处理』
第 522 页的『execute』
第 544 页的『get next』
第 549 页的『get previous』
第 490 页的『I/O 错误值』
第 561 页的『open』
第 574 页的『prepare』
第 576 页的『replace』
第 61 页的『SQL 项属性』
第 860 页的『terminalID』

get absolute

EGL **get absolute** 语句读取关系数据库结果集中用数字指定的行。将从结果集的开头（如果指定正数）或结果集的结尾（如果指定负数）标识该行。

仅当在相关 **open** 语句中指定了 **scroll** 选项时，才能使用此语句。



position

整数项或文字。

如果 *position* 的值是正数，将从结果集开头指定该行。例如，指定 **get absolute 1** 将检索第一行并且相当于指定 **get first**。指定 **get absolute 2** 将检索第二行。

如果 *position* 的值是负数，将从结果集结尾指定该行。例如，指定 **get absolute -1** 将检索最后一行并且相当于指定 **get last**。指定 **get absolute -2** 将检索倒数第二行。

position 的值为零时将导致硬错误，如异常处理中所述。

record name

SQL 记录的名称。

from resultSetID

这是一个标识，它使 **get absolute** 语句与同一程序中之前运行的 **open** 语句相关。有关详细信息，请参阅 *resultSetID*。

into

开始 EGL **into** 子句，该子句列示从关系数据库表接收值的项。

item

用于接收特定列的值的项。不要在项名前面加冒号 (:)。

如果发出 **get absolute** 语句以检索由带有 **forUpdate** 选项的 **open** 语句选择的行，则可以执行下列任何操作：

- 通过 EGL **replace** 语句更改行
- 通过 EGL **delete** 语句除去行
- 通过 EGL **execute** 语句更改或除去行

SQL **FETCH** 语句表示生成的代码中的 EGL **get absolute** 语句。生成的 SQL 语句的格式不能更改，但设置 **INTO** 子句除外。

如果发出 **get absolute** 语句并且该语句尝试访问不在结果集中的行，EGL 运行时将进行如下操作：

- 不会从结果集中复制数据
- 让游标处于打开状态，并且游标位置保持不变
- 将 SQL 记录（如果有的话）设置为 **noRecordFound**

一般来说，如果发生了错误并且处理继续进行，游标将保持打开状态，并且游标位置保持不变。

最后，当指定 SQL COMMIT 或 sysLib.commit 时，代码保留 **open** 语句中声明的游标中的位置，但仅当在 **open** 语句中使用 hold 选项时才会如此。

相关概念

第 678 页的『resultSetID』

第 209 页的『SQL 支持』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 520 页的『delete』

第 86 页的『异常处理』

第 522 页的『execute』

第 533 页的『get』

『get current』

第 541 页的『get first』

第 543 页的『get last』

第 544 页的『get next』

第 549 页的『get previous』

第 552 页的『get relative』

第 80 页的『EGL 语句』

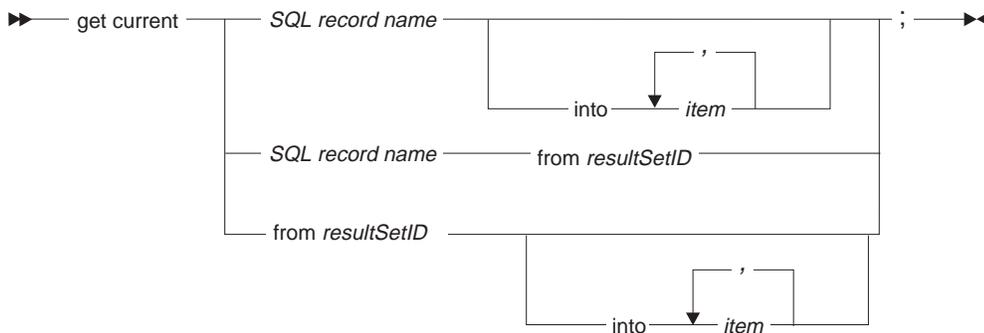
第 561 页的『open』

第 576 页的『replace』

get current

EGL **get current** 语句读取关系数据库结果集中游标已定位的行。

仅当在相关 **open** 语句中指定了 scroll 选项时，才能使用此语句。



record name

SQL 记录的名称。

from *resultSetID*

这是一个标识，它使 **get current** 语句与同一程序中之前运行的 **open** 语句相关。有关详细信息，请参阅 *resultSetID*。

into

开始 EGL **into** 子句，该子句列示从关系数据库表接收值的项。

item

用于接收特定列的值的项。不要在项名前面加冒号 (:)。

如果发出 **get current** 语句以检索由带有 **forUpdate** 选项的 **open** 语句选择的行，则可以执行下列任何操作：

- 通过 EGL **replace** 语句更改行
- 通过 EGL **delete** 语句除去行
- 通过 EGL **execute** 语句更改或除去行

SQL **FETCH** 语句表示生成的代码中的 EGL **get current** 语句。生成的 SQL 语句的格式不能更改，但设置 **INTO** 子句除外。

如果发生了错误并且处理继续进行，游标将保持打开状态。

最后，当指定 **SQL COMMIT** 或 `sysLib.commit` 时，代码保留 **open** 语句中声明的游标中的位置，但仅当在 **open** 语句中使用 **hold** 选项时才会如此。

相关概念

第 678 页的『*resultSetID*』

第 209 页的『SQL 支持』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 520 页的『*delete*』

第 522 页的『*execute*』

第 533 页的『*get*』

第 538 页的『*get absolute*』

『*get first*』

第 543 页的『*get last*』

第 544 页的『*get next*』

第 549 页的『*get previous*』

第 552 页的『*get relative*』

第 80 页的『EGL 语句』

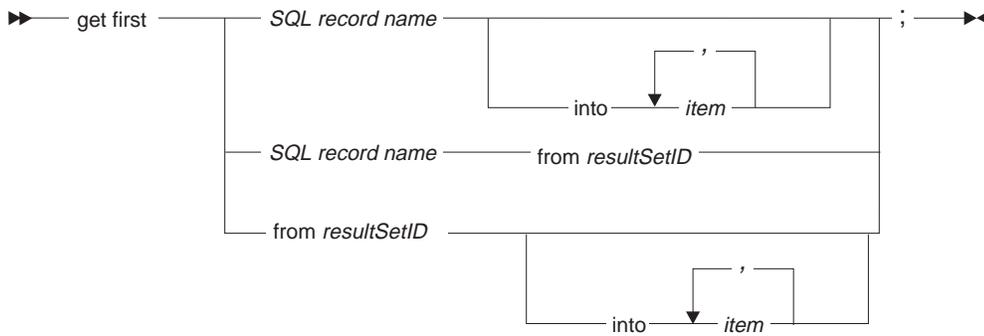
第 561 页的『*open*』

第 576 页的『*replace*』

get first

EGL **get first** 语句读取关系数据库结果集中的第一行。

仅当在相关 **open** 语句中指定了 **scroll** 选项时，才能使用此语句。



record name

SQL 记录的名称。

from *resultSetID*

这是一个标识，它使 **get first** 语句与同一程序中之前运行的 **open** 语句相关。有关详细信息，请参阅 *resultSetID*。

into

开始 EGL **into** 子句，该子句列示从关系数据库表接收值的项。

item

用于接收特定列的值的项。不要在项名前面加冒号 (:)。

如果发出 **get first** 语句以检索由带有 **forUpdate** 选项的 **open** 语句选择的行，则可以执行下列任何操作：

- 通过 EGL **replace** 语句更改行
- 通过 EGL **delete** 语句除去行
- 通过 EGL **execute** 语句更改或除去行

SQL **FETCH** 语句表示生成的代码中的 EGL **get first** 语句。生成的 SQL 语句的格式不能更改，但设置 **INTO** 子句除外。

如果发生了错误并且处理继续进行，游标将保持打开状态。

最后，当指定 **SQL COMMIT** 或 `sysLib.commit` 时，代码保留 **open** 语句中声明的游标中的位置，但仅当在 **open** 语句中使用 **hold** 选项时才会如此。

相关概念

第 678 页的『*resultSetID*』

第 209 页的『SQL 支持』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 520 页的『*delete*』

第 522 页的『*execute*』

第 533 页的『*get*』

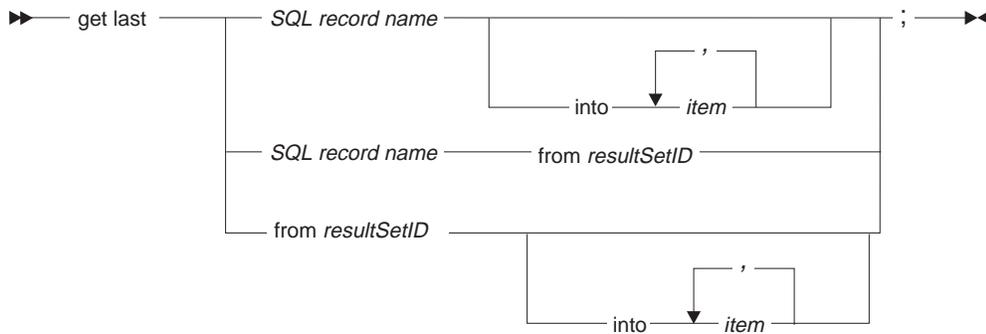
第 538 页的『*get absolute*』

第 540 页的『 get current 』
『 get last 』
第 544 页的『 get next 』
第 549 页的『 get previous 』
第 552 页的『 get relative 』
第 80 页的『 EGL 语句 』
第 561 页的『 open 』
第 576 页的『 replace 』

get last

EGL **get last** 语句读取关系数据库结果集中的最后一行。

仅当在相关 **open** 语句中指定了 **scroll** 选项时，才能使用此语句。



record name

SQL 记录的名称。

from *resultSetID*

这是一个标识，它使 **get last** 语句与同一程序中之前运行的 **open** 语句相关。有关详细信息，请参阅 *resultSetID*。

into

开始 EGL **into** 子句，该子句列示从关系数据库表接收值的项。

item

用于接收特定列的值的项。不要在项名前面加冒号 (:)。

如果发出 **get last** 语句以检索由带有 **forUpdate** 选项的 **open** 语句选择的行，则可以执行下列任何操作：

- 通过 EGL **replace** 语句更改行
- 通过 EGL **delete** 语句除去行
- 通过 EGL **execute** 语句更改或除去行

SQL **FETCH** 语句表示生成的代码中的 EGL **get last** 语句。生成的 SQL 语句的格式不能更改，但设置 **INTO** 子句除外。

如果发生了错误并且处理继续进行，游标将保持打开状态。

最后，当指定 SQL COMMIT 或 sysLib.commit 时，代码保留 **open** 语句中声明的游标中的位置，但仅当在 **open** 语句中使用 hold 选项时才会如此。

相关概念

第 678 页的『resultSetID』

第 209 页的『SQL 支持』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 520 页的『delete』

第 522 页的『execute』

第 533 页的『get』

第 538 页的『get absolute』

第 540 页的『get current』

第 541 页的『get first』

『get next』

第 549 页的『get previous』

第 552 页的『get relative』

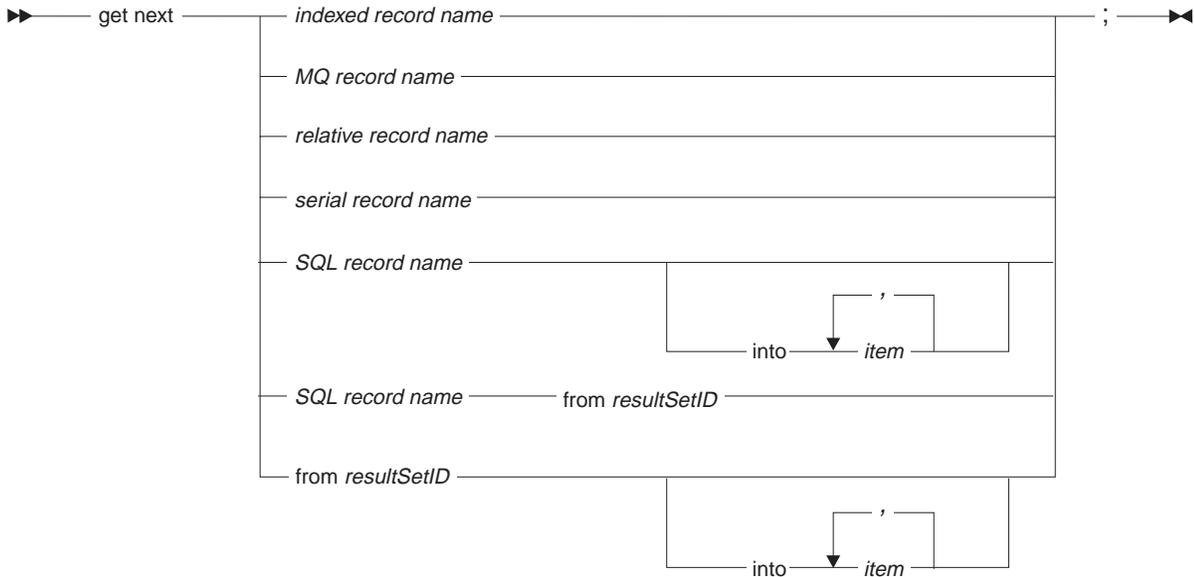
第 80 页的『EGL 语句』

第 561 页的『open』

第 576 页的『replace』

get next

EGL **get next** 语句从文件或消息队列中读取下一条记录，或者从数据库中读取下一行。



record name

I/O 对象的名称: 带索引记录、MQ 记录、相对记录、串行记录或 SQL 记录。

from *resultSetID*

(仅适用于 SQL 处理) 这是一个标识, 它使 **get next** 语句与同一程序中之前运行的 **open** 语句相关。有关详细信息, 请参阅 *resultSetID*。

into

开始 EGL **into** 子句, 该子句列示从关系数据库表接收值的项。

item

用于接收特定列的值的项。不要在项名前面加冒号 (:)。

下面是文件访问示例:

```
try
  open record1 forUpdate;
onException
  myErrorHandler(8);
return;
end

try
  get next record1;
onException
  myErrorHandler(12);
return;
end

while (record1 not endOfFile)
  makeChanges(record1); // process the record

  try
    replace record1;
onException
  myErrorHandler(16);
return;
end

try
  get next record1;
onException
  myErrorHandler(12);
return;
end

end // end while

sysLib.commit();
```

有关 **get next** 语句的详细信息取决于记录类型。有关 SQL 处理的详细信息, 请参阅第 548 页的『SQL 处理』。

带索引记录

当对带索引记录执行 **get next** 语句时, 其作用取决于当前文件位置, 该位置是由下列任何一项操作设置的:

- 成功的输入或输出 (I/O) 操作, 如 **get** 语句或另一个 **get next** 语句; 或者
- *set record position* 格式的 **set** 语句。

规则如下所示:

- 当文件未打开时, **get next** 语句读取文件中具有最小键值的记录。

- 每个后续 **get next** 语句都将读取相对于当前文件位置具有下一个最大键值的记录。重复键的例外情况在后面描述。
- 在 **get next** 语句读取文件中具有最大键值的记录之后，下一个 **get next** 语句将导致 I/O 错误值 **endOfFile**。
- 当前文件位置会受下列任何操作的影响：
 - *set record position* 格式的 EGL **set** 语句根据 *set value*（这是由 **set** 语句引用的带索引记录中的键值）确定文件位置。对同一个带索引记录执行的后续 **get next** 语句将读取键值等于或大于 *set value* 的文件记录。如果不存在这样的记录，则 **get next** 的结果是 **endOfFile**。
 - 除 **get next** 语句之外的成功 I/O 语句将确定新的文件位置，并且，对同一个 EGL 记录发出的后续 **get next** 语句将读取下一个文件记录。例如，在 **get previous** 语句读取文件记录之后，**get next** 语句或者读取具有下一个最大键的文件记录，或者返回 **endOfFile**。
 - 如果 **get previous** 语句返回 **endOfFile**，则后续 **get next** 语句检索文件中的第一条记录。
 - 在不成功的 **get**、**get next** 或 **get previous** 语句之后，文件位置是未定义的并且必须由 *set record position* 格式的 **set** 语句重新确定，或者由除 **get next** 或 **get previous** 语句之外的 I/O 操作来重新确定。
- 当您正在使用备用索引且在文件中存在重复键时，下列规则适用：
 - 只有在 **get next** 语句读取了所有与最近检索的记录具有相同的键的记录之后，才检索具有更大键值的记录。键重复的记录的检索顺序就是 VSAM 返回记录的顺序。
 - 如果在除 **get next** 之外的成功 I/O 操作之后执行 **get next**，则 **get next** 将跳过任何具有重复键的记录，并检索具有下一个更大键的记录。
 - 当程序检索包含同一个键的一组记录中的最后一个记录时，不设置 EGL 错误值 **duplicate**。

考虑包含下列键的文件：

1, 2, 2, 2, 3, 4

下列每一个表都说明对同一带索引记录运行一系列 EGL 语句的效果。

EGL 语句（按顺序）	带索引记录中的键	语句检索到的文件记录中的键	EGL 错误值
get	2	2（三个中的第一个）	duplicate
get next	任何	2（第二个）	duplicate
get next	任何	2（第三个）	—
get next	任何	3	—

EGL 语句（按顺序）	带索引记录中的键	语句检索到的文件记录中的键	EGL 错误值
set （具有 <i>set record position</i> 格式）	2	不检索	duplicate
get next	任何	2（三个中的第一个）	—

EGL 语句 (按顺序)	带索引记录中的键	语句检索到的文件记录中的键	EGL 错误值
get next	任何	2 (第二个)	duplicate
get next	任何	2 (第三个)	—
get next	任何	3	—

消息队列

当对 MQ 记录执行 **get next** 时，将把队列中的第一条记录读入 MQ 记录。进行这种安排的原因是 **get next** 调用了一个或多个 MQSeries 调用：

- MQCONN 将生成的代码与缺省队列管理器连接，并且在无任何活动连接时被调用
- MQOPEN 建立与队列的连接并且在存在活动连接但队列未打开时被调用
- MQGET 从队列中除去记录，除非在先前的 MQSeries 调用中发生了错误，否则总是会调用 MQGET

相对记录

当对相对记录执行 **get next** 语句时，作用取决于当前文件位置，该位置是由成功的输入或输出 (I/O) 操作 (如 **get** 语句或另一个 **get next** 语句) 设置的。规则如下所示：

- 当文件未打开时，**get next** 语句读取文件中的第一条记录。
- 每个后续 **get next** 都将读取相对于当前文件位置具有下一个最大键值的记录。
- 如果下一条记录已被删除，则 **get next** 不会返回 **noRecordFound**。**get next** 将跳过已删除的记录并检索文件中的下一条记录。
- 在 **get next** 语句读取文件中具有最大键值的记录之后，下一个 **get next** 语句将导致 EGL 错误值 **endOfFile**。
- 当前文件位置会受下列任何操作的影响：
 - 除 **get next** 之外的成功 I/O 语句将确定新的文件位置，并且，对同一个 EGL 记录发出的后续 **get next** 将读取下一个文件记录。
 - 在不成功的 **get**、**get next** 或 **get previous** 语句之后，文件位置是未定义的并且必须由 *set record position* 格式的 **set** 语句重新确定，或者由除 **get next** 语句之外的 I/O 操作来重新确定。
- 在 **get next** 语句读取文件中最后一条记录之后，下一个 **get next** 语句将导致 EGL 错误值 **endOfFile** 和 **noRecordFound**。

串行记录

当对串行记录执行 **get next** 语句时，其作用取决于当前文件位置，该位置是由另一个 **get next** 语句设置的。规则如下所示：

- 当文件未打开时，**get next** 语句读取文件中的第一条记录。
- 每个后续 **get next** 语句都读取下一条记录。
- 在 **get next** 语句读取了最后一条记录之后，后续的 **get next** 语句将导致 EGL 错误值 **endOfFile**。
- 如果生成的代码添加一个串行记录，然后对同一个文件发出与 **get next** 语句具有等同作用的语句，则在执行 **get next** 语句之前，EGL 将关闭并重新打开该文件。因

此，跟随在 **add** 语句之后的 **get next** 语句读取文件中的第一条记录。当 **get next** 和 **add** 语句位于不同的程序中，并且其中一个程序调用另一个程序时，也会发生这种行为。

建议避免在多个程序中同时打开同一个文件。

SQL 处理

当对 SQL 记录执行 **get next** 语句时，代码将从 **open** 语句选择的那些行中读取下一行。如果发出 **get next** 语句以检索由带有 **forUpdate** 选项的 **open** 语句选择的行，则可以执行下列任何操作：

- 通过 EGL **replace** 语句更改行
- 通过 EGL **delete** 语句除去行
- 通过 EGL **execute** 语句更改或除去行

SQL **FETCH** 语句表示生成的代码中的 EGL **get next** 语句。生成的 SQL 语句的格式不能更改，但设置 **INTO** 子句除外。

如果发出 **get next** 语句并且该语句尝试访问位于最后一个所选行之后的行，下列描述是适用的：

- 不会从结果集中复制任何数据
- EGL 将 SQL 记录（如果有的话）设置为 **noRecordFound**
- 如果相关的 **open** 语句包括 **scroll** 选项，则游标将保持打开状态，并且游标的位置保持不变。输出时，**scroll** 选项才有效。
- 如果未设置 **scroll** 选项，游标将会关闭。

最后，当指定 SQL **COMMIT** 或 **sysLib.commit** 时，代码保留 **open** 语句中声明的游标中的位置，但仅当在 **open** 语句中使用 **hold** 选项时才会如此。

相关概念

第 125 页的『记录类型和属性』

第 678 页的『resultSetID』

第 209 页的『SQL 支持』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 511 页的『add』

第 517 页的『close』

第 520 页的『delete』

第 86 页的『异常处理』

第 522 页的『execute』

第 533 页的『get』

第 549 页的『get previous』

第 490 页的『I/O 错误值』

第 80 页的『EGL 语句』

第 561 页的『open』

第 574 页的『prepare』

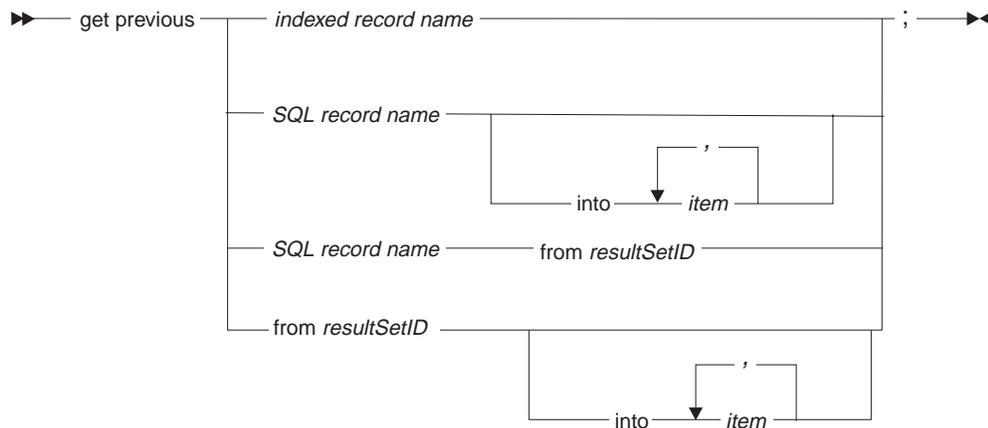
第 576 页的『replace』

第 579 页的『set』

get previous

EGL **get previous** 语句读取关系数据库结果集中的上一行或文件中的上一个记录，该文件与指定 EGL 的带索引记录相关联。

仅当在相关 **open** 语句中指定了 **scroll** 选项时，才能对关系数据库结果集使用此语句。



record name

I/O 对象的名称：带索引记录或 SQL 记录。

from resultSetID

（仅适用于 SQL 处理）这是一个标识，它使 **get previous** 语句与同一程序之前运行的 **open** 语句相关。有关详细信息，请参阅 *resultSetID*。

into

开始 EGL into 子句，该子句列示从关系数据库表接收值的项。

item

用于接收特定列的值的项。不要在项名前面加冒号（:）。

下面是带索引记录的示例：

```
record1.hexKey = "FF";
set record1 position;

try
  get previous record1;
onException
  myErrorHandler(8);
return;
end

while (record1 not endOfFile)
  processRecord(record1); // handle the data

try
  get previous record1;
onException
```

```
        myErrorHandler(8);
        return;
    end

end
```

有关 **get previous** 的详细信息取决于您是否在使用带索引记录或是否与第 552 页的『SQL 处理』有关。

带索引记录

当对带索引记录执行 **get previous** 语句时，其作用取决于当前文件位置，该位置是由以下操作之一设置的：

- 成功的输入或输出（I/O）操作，如 **get** 语句或另一个 **get previous** 语句；或者
- *set record position* 格式的 **set** 语句。

带索引记录的规则如下所示：

- 当文件未打开时，**get previous** 语句读取文件中具有最大键值的记录。
- 每个后续 **get previous** 都将读取相对于当前文件位置具有下一个最小键值的记录。重复键的例外情况在后面描述。
- 在 **get previous** 语句读取文件中具有最小键值的记录之后，下一个 **get previous** 语句将导致 EGL 错误值 **endOfFile**。
- 当前文件位置会受下列任何操作的影响：
 - *set record position* 格式的 EGL **set** 语句根据 *set value*（这是由 **set** 语句引用的带索引记录中的键值）确定文件位置。对同一个带索引记录执行的后续 **get previous** 语句将读取键值等于或小于 *set value* 的文件记录。如果不存在这样的记录，则 **get previous** 语句的结果是 **endOfFile**。

如果用十六进制 FF 填充了 *set value*，则 *set record position* 格式的 **set** 语句的结果如下所示：

- **set** 语句在文件中的最后一条记录之后确定文件位置
 - 如果下一个 I/O 操作是 **get previous** 语句，则生成的代码检索文件中的最后一条记录
 - 除 **get previous** 语句之外的成功 I/O 语句将确定新的文件位置，并且，对同一个 EGL 记录发出的后续 **get previous** 语句将读取上一个文件记录。例如，在 **get next** 语句读取文件记录之后，**get previous** 语句或者读取具有下一个最小键的文件记录，或者返回 **endOfFile**。
 - 如果 **get next** 语句返回 **endOfFile**，则后续 **get previous** 语句检索文件中的最后一条记录。
 - 在不成功的 **get**、**get next** 或 **get previous** 语句之后，文件位置是未定义的并且必须由 *set record position* 格式的 **set** 语句重新确定，或者由除 **get next** 或 **get previous** 语句之外的 I/O 操作来重新确定。
- 当您正在使用备用索引且在文件中存在重复键时，下列规则适用：
 - 只有在 **get previous** 语句读取了所有与最近检索的记录具有相同的键的记录之后，才检索具有更小键值的记录。键重复的记录的检索顺序就是 VSAM 返回记录的顺序。

- 如果在除 **get previous** 之外的成功 I/O 操作之后执行 **get previous** 语句，则 **get previous** 语句将跳过任何具有重复键的记录，并检索具有下一个更小键的记录。
- 当程序检索包含同一个键的一组记录中的最后一个记录时，不设置 EGL 错误值 **duplicate**。

考虑这样一个文件，在该文件中，备用索引中的各个键如下所示：

1, 2, 2, 2, 3, 4

下列每一个表都说明对同一带索引记录运行一系列 EGL 语句的效果。

EGL 语句 (按顺序)	带索引记录中的键	语句检索到的文件记录中的键	COBOL 的 EGL 错误值
set (具有 <i>set record position</i> 格式)	1	--	--
get previous	任何	1	--

EGL 语句 (按顺序)	带索引记录中的键	语句检索到的文件记录中的键	EGL 错误值
get	3	3	—
get previous	任何	2 (三个中的第一个)	duplicate
get previous	任何	2 (第二个)	duplicate
get previous	任何	2 (第三个)	—
get previous	任何	1	—

EGL 语句 (按顺序)	带索引记录中的键	语句检索到的文件记录中的键	EGL 错误值
set (具有 <i>set record position</i> 格式)	2	—	duplicate
get next	任何	2 (第一个)	—
get next	任何	2 (第二个)	duplicate
get previous	任何	1	—
get previous	任何	--	endOfFile

EGL 语句 (按顺序)	带索引记录中的键	语句检索到的文件记录中的键	EGL 错误值
set (具有 <i>set record position</i> 格式)	1	--	--
get previous	任何	1	--

SQL 处理

当对 SQL 记录执行 **get previous** 语句时，代码将从 **open** 语句选择的那些行读取上一行，但仅当您指定了 **scroll** 选项才会如此。如果发出 **get previous** 语句以检索由同时带有 **forUpdate** 选项的 **open** 语句选择的行，可以执行下列任何任务：

- 通过 EGL **replace** 语句更改行
- 通过 EGL **delete** 语句除去行
- 通过 EGL **execute** 语句更改或除去行

SQL **FETCH** 语句表示生成的代码中的 EGL **get previous** 语句。生成的 SQL 语句的格式不能更改，但设置 **INTO** 子句除外。

如果发出 **get previous** 语句并且该语句尝试访问位于第一个所选行之前的行，EGL 运行时将进行如下操作：

- 不会从结果集中复制数据
- 让游标处于打开状态，并且游标位置保持不变
- 将 SQL 记录（如果有的话）设置为 **noRecordFound**

一般来说，如果发生了错误并且处理继续进行，游标将保持打开状态，并且游标位置保持不变。

最后，当指定 SQL **COMMIT** 或 `sysLib.commit` 时，代码保留 **open** 语句中声明的游标中的位置，但仅当在 **open** 语句中使用 **hold** 选项时才会如此。

相关概念

第 125 页的『记录类型和属性』

相关任务

第 690 页的『EGL 语句和命令的语法图』

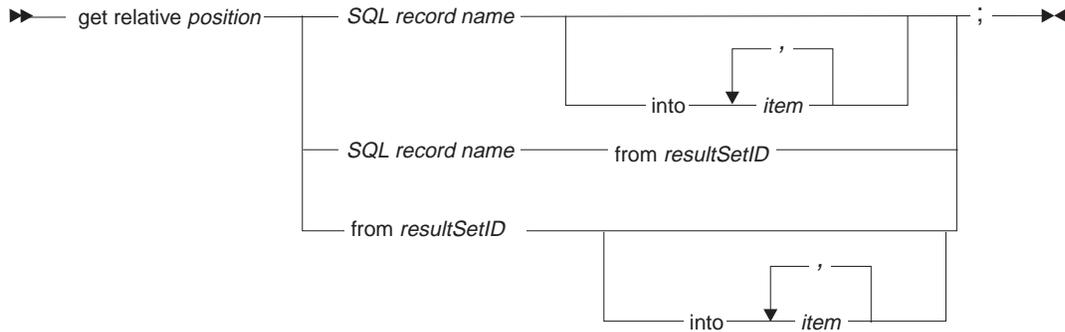
相关参考

- 第 511 页的『**add**』
- 第 517 页的『**close**』
- 第 520 页的『**delete**』
- 第 86 页的『异常处理』
- 第 522 页的『**execute**』
- 第 533 页的『**get**』
- 第 544 页的『**get next**』
- 第 490 页的『I/O 错误值』
- 第 561 页的『**open**』
- 第 574 页的『**prepare**』
- 第 80 页的『EGL 语句』
- 第 576 页的『**replace**』
- 第 579 页的『**set**』

get relative

EGL **get relative** 语句读取关系数据库结果集中用数字指定的行。该行是根据结果集中的游标位置指定的。

仅当在相关 **open** 语句中指定了 **scroll** 选项时，才能使用此语句。



position

整数项或文字。

如果 *position* 的值是正数，则该位置是结果集中当前数字位置的增量。例如，当游标在第一行上时指定 **get relative 2** 将检索第三行；指定 **get relative 1** 相当于指定 **get next**。

如果 *position* 的值是负数，则该位置是结果集中当前数字位置的减量。例如，当游标在第三行上时指定 **get relative -2** 将检索第一行；指定 **get relative -1** 相当于指定 **get previous**。

如果 *position* 的值为零，则将检索已经起作用的游标位置所在的行并且相当于指定 **get current**。

record name

SQL 记录的名称。

from resultSetID

这是一个标识，它使 **get relative** 语句与同一程序中之前运行的 **open** 语句相关。有关详细信息，请参阅 *resultSetID*。

into

开始 EGL **into** 子句，该子句列示从关系数据库表接收值的项。

item

用于接收特定列的值的项。不要在项名前面加冒号 (:)。

如果发出 **get relative** 语句以检索由带有 **forUpdate** 选项的 **open** 语句选择的行，则可以执行下列任何操作：

- 通过 EGL **replace** 语句更改行
- 通过 EGL **delete** 语句除去行
- 通过 EGL **execute** 语句更改或除去行

SQL **FETCH** 语句表示生成的代码中的 EGL **get relative** 语句。生成的 SQL 语句的格式不能更改，但设置 **INTO** 子句除外。

如果发出 **get relative** 语句并且该语句尝试访问不在结果集中的行，EGL 运行时将进行如下操作：

- 不会从结果集中复制数据
- 让游标处于打开状态，并且游标位置保持不变
- 将 SQL 记录（如果有的话）设置为 **noRecordFound**

一般来说，如果发生了错误并且处理继续进行，游标将保持打开状态，并且游标位置保持不变。

最后，当指定 SQL COMMIT 或 sysLib.commit 时，代码保留 **open** 语句中声明的游标中的位置，但仅当在 **open** 语句中使用 hold 选项时才会如此。

相关概念

第 678 页的『resultSetID』

第 209 页的『SQL 支持』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 520 页的『delete』

第 86 页的『异常处理』

第 522 页的『execute』

第 533 页的『get』

第 538 页的『get absolute』

第 540 页的『get current』

第 541 页的『get first』

第 543 页的『get last』

第 544 页的『get next』

第 549 页的『get previous』

第 80 页的『EGL 语句』

第 561 页的『open』

第 576 页的『replace』

goTo

EGL **goTo** 语句导致处理在指定的标号处继续，该标号必须与该语句位于同一函数中并且位于块外部。

▶—goto *label* : —————▶

label

在函数中的别处显示的一系列字符，它位于任何块外部，这些块包括：

- if
- else
- when（位于 **case** 语句中）
- while
- try

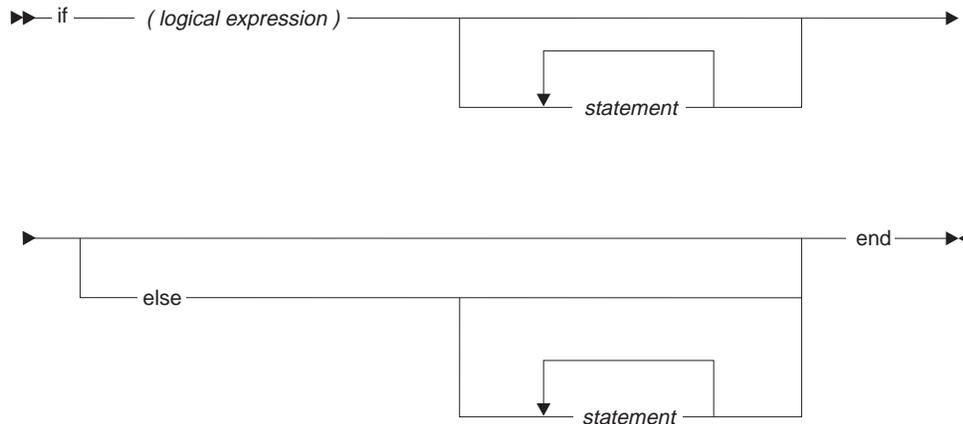
当在继续进行处理的位置显示时，标号的后面有一个冒号。有关标号的有效字符的详细信息，请参阅命名约定。

相关参考

第 612 页的『命名约定』

if, else

EGL 关键字 **if** 标记一组语句（如果有的话）的开始，仅当逻辑表达式解析为 **true** 时才会运行该组语句。可选关键字 **else** 标记备用语句组（如果有的话）的开始，仅当逻辑表达式解析为 **false** 时才会运行该组语句。关键字 **end** 标记 **if** 语句的结束。



logical expression

得出的值为 **true** 或 **false** 的表达式（一系列操作数和运算符）

statement

一个或多个 EGL 语句

可将 **if** 和其它以 **end** 终止的语句嵌套至任何级别。每个 **end** 关键字都对应于以下列其中一个关键字开始并且未结束的最近语句：

- **if**
- **case**
- **try**
- **while**

那些语句的后面都不跟随分号。

下面是一个示例：

```
if (userRequest == "U")
  try
    update myRecord;
    onException
      myErrorHandler(12); // ends program
  end
  try
    myRecord.myItem=25;
    replace record1;
    onException
      myErrorHandler(16);
  end
end
```

```

else
  try
    add record2;
    onException
      myErrorHandler(18); // ends program
    end
    if (sysVar.systemType is WIN)
      myFunction01();
    else
      myFunction02();
    end
  end
end

```

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 454 页的『逻辑表达式』

第 80 页的『EGL 语句』

move

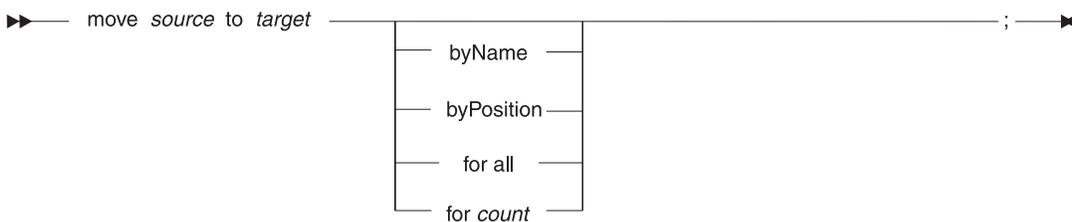
EGL **move** 语句以下列三种方式中的任何一种复制数据。第一种选择是逐个字节复制数据；第二种选择（称为按名称）将数据从一个结构中的命名字段复制至另一个结构中的同名字段；第三种选择（称为按位置）将一个结构中每个字段的数据复制至另一个结构中处于同等位置的字段。

下列一般规则适用：

- 如果源值是下列其中之一，则缺省选择是逐个字节复制数据：
 - 基本变量
 - 固定结构中的字段
 - 文字
 - 常量

否则，缺省选择是按名称复制数据。

- 对于 **move**，将检查字段与字段间的兼容性。**move** 语句截断、填充和类型转换的规则与 **assignment** 语句相同，但是它们的整体行为不同。
- 当使用动态数组时，最后一个元素由数组的当前大小确定。**move** 语句决不会将元素添加至数组；要扩展动态数组，使用特定于数组的函数 `appendElement` 或 `appendAll`，如数组中所述。



通过参考下列类别，可以很好地理解该语句：

byName

在指定 **byName** 时，数据将从源中的每个字段写至目标中的同名字段。操作按照各个字段在源中的顺序进行。

源和目标可为如下所示：

source

下列其中一项：

- 固定记录的动态数组；但仅当目标不是记录时，该数组才有效
- 记录
- 固定记录
- 带有子结构的结构字段
- 带有子结构的结构字段数组；但仅当目标不是记录时，此数组才有效
- `dataTable`
- 表单

名称为星号 (*) 的固定结构字段不是作为源字段提供的，但该字段的子结构中的所有命名字段都可用。

target

下列其中一项：

- 固定记录的动态数组；但仅当源不是记录时，此数组才有效
- 记录
- 固定记录
- 带有子结构的结构字段
- 带有子结构的结构字段数组；但仅当源不是记录时，此数组才有效
- `dataTable`
- 表单
-

示例语句如下所示：

```
move myRecord01 to myRecord02 byName;
```

在下列任何情况下，该操作无效：

- 源中的两个或多个字段具有相同名称；
- 目标中的两个或多个字段具有相同名称；
- 源字段是多维结构字段数组或其容器是数组的一维结构字段数组；或者
- 目标字段是多维结构字段数组或其容器是数组的一维结构字段数组。

在下列情况下，该操作有效：

- 在很简单的情况下，源是固定结构但它本身不是数组元素，目标也是一样。下列规则适用：
 - 如果未调用任何数组，则源结构中的每个下级字段的值将复制至目标结构中的同名字段。
 - 如果结构字段数组将复制至结构字段数组，则该操作被视为全部移动：
 - 源字段的元素将复制至目标字段的连续元素

- 如果源数组的元素少于目标数组的元素，则在复制源数组的最后一个元素后处理会停止
- 在另一情况下，源或目标是记录。源中的字段将被指定给目标中的同名字段。
- 较复杂的情况用示例介绍。源是包含 10 个固定记录的数组，每个固定记录包括下列结构字段：

```
10 empnum CHAR(3);
10 empname CHAR(20);
```

目标是包括下列结构字段的固定结构：

```
10 empnum CHAR(3)[10];
10 empname CHAR(20)[10];
```

该操作将第一个固定记录中的字段 `empnum` 的值复制至结构字段数组 `empnum` 的第一个元素；将第一个固定记录中的字段 `empname` 的值复制至结构字段数组 `empname` 的第一个元素；并且对源数组中的每个固定记录执行同样的操作。

如果源是具有类似如下的子结构的单个固定记录，则会进行等效的操作：

```
10 mySubStructure[10]
  15 empnum CHAR(3);
  15 empname CHAR(20);
```

- 最后，考虑源是包括下列结构字段的固定记录的情况：

```
10 empnum CHAR(3);
10 empname CHAR(20)[10];
```

目标是具有以下子结构的表单、固定记录或结构字段：

```
10 empnum char(3)[10];
10 empname char(20);
```

将字段 `empnum` 的值从源复制至目标中的 `empnum` 的第一个元素；将 `empname` 的第一个元素的值从源复制至目标中的字段 `empname`。

byPosition

byPosition 的用途是将一个结构中每个字段的数据复制至另一个结构中同等位置的字段。

源和目标可为如下所示：

source

下列其中一项：

- 固定记录的动态数组；但仅当目标不是记录时，该数组才有效
- 记录
- 固定记录
- 带有子结构的结构字段
- 带有子结构的结构字段数组；但仅当目标不是记录时，此数组才有效
- `dataTable`
- 表单

target

下列其中一项：

- 固定记录的动态数组；但仅当源不是记录时，此数组才有效

- 记录
- 固定记录
- 带有子结构的结构字段
- 带有子结构的结构字段数组； 但仅当源不是记录时， 此数组才有效
- dataTable
- 表单

在记录与固定结构间移动数据时， 只考虑固定结构的顶层字段。 在两个固定结构间移动数据时， 只考虑每个结构的最底层（叶）字段。

如果源或目标字段是多维结构字段数组或其容器是数组的一维结构字段数组， 则该操作无效。

在下列情况下， 该操作有效：

- 在很简单的情况下， 源是固定结构但它本身不是数组元素， 目标也是一样。 下列规则适用：
 - 如果未调用任何数组， 则源结构中的每个叶字段的值将复制至目标结构中相应位置的叶字段。
 - 如果结构字段数组将复制至结构字段数组， 则该操作被视为全部移动：
 - 源字段的元素将复制至目标字段的连续元素
 - 如果源数组的元素少于目标数组的元素， 则在复制源数组的最后一个元素后处理会停止
- 在另一情况下， 源或目标是记录。 源的顶层或叶字段（根据源类型）将指定给目标中的顶层或叶字段（根据目标类型）。
- 较复杂的情况用示例介绍。 源是包含 10 个固定记录的数组， 每个固定记录包括下列结构字段：

```
10 empnum CHAR(3);
10 empname CHAR(20);
```

目标是包括下列结构字段的固定结构：

```
10 empnum CHAR(3)[10];
10 empname CHAR(20)[10];
```

该操作将第一个固定记录中的字段 empnum 的值复制至结构字段数组 empnum 的第一个元素； 将第一个固定记录中的字段 empname 的值复制至结构字段数组 empname 的第一个元素； 并且对源数组中的每个固定记录执行同样的操作。

如果源是具有类似如下的子结构的单个固定记录， 则会进行等效的操作：

```
10 mySubStructure[10]
  15 empnum CHAR(3);
  15 empname CHAR(20);
```

- 最后， 考虑源是包括下列结构字段的固定记录的情况：

```
10 empnum CHAR(3);
10 empname CHAR(20)[10];
```

目标是具有以下子结构的表单、 固定记录或结构字段：

```
10 empnum char(3)[10];
10 empname char(20);
```

将字段 `empnum` 的值从源复制至目标中的 `empnum` 的第一个元素；将 `empname` 的第一个元素的值从源复制至目标中的字段 `empname`。

for all

for all 的用途是对目标数组中的所有元素赋值。

源和目标可为如下所示：

source

下列其中一项：

- 记录、固定记录或基本变量的动态数组
- 记录
- 固定记录
- 带有子结构或不带子结构的结构字段
- 带有子结构或不带子结构的结构字段数组
- 基本变量
- 文字或常量

target

下列其中一项：

- 记录、固定记录或基本变量的动态数组
- 带有子结构或不带子结构的结构字段数组
- 动态数组或结构字段数组的元素

此情况下的 **move** 语句相当于多个 **assignment** 语句（每个目标数组元素一个赋值语句），如果尝试赋值无效，则可能会发生错误。有关有效性的详细信息，请参阅 [赋值](#)。

如果源或目标元素具有固定结构，则 **move** 语句会将该结构视作类型为 `CHAR` 的字段，除非结构的顶层指定了另一基本类型。在使用 **for all** 时，**move** 语句不考虑子结构。

如果源是数组的元素，则源将被视作一个数组，其中的指定元素是第一个元素，先前的元素将被忽略。

如果源是数组或数组的元素，则源数组的每个连续元素将复制至目标数组中按顺序进行的下一个元素。目标数组或源数组可能较长，在另一数组中具有匹配元素的最后一个元素中的数据复制完毕后，操作就结束了。

如果源既不是数组也不是数组元素，则该操作会使用源值来初始化目标数组的每个元素。

for count

for count 的用途是对目标数组中的元素的顺序子集赋值。示例如下所示：

- 下面的语句将“abc”移至目标中的元素 7、8 和 9。

```
move "abc" to target[7] for 3
```

- 下面的语句将元素 2、3 和 4 从源移至目标中的元素 7、8 和 9。

```
move source[2] to target[7] for 3
```

在下列情况下，该操作有效：

- 如果源既不是数组也不是数组元素，则该操作会使用源值来初始化目标数组的元素。
- 如果源是数组，则该数组的第一个元素将是要复制的一组元素中的第一项。如果源是数组的元素，则该元素将是要复制的一组元素中的第一项。
- 如果目标是数组，则该数组的第一个元素将是要接收数据的一组元素中的第一项。如果目标是数组的元素，则该元素将是要接收数据的一组元素中的第一项。

count 值指示要接收数据的目标元素的数目。该值可以是下列其中任何一项：

- 整数文字
- 解析为整数的变量
- 数字表达式，但并非函数调用

move 语句相当于多个 **assignment** 语句（每个目标数组元素一个赋值语句），如果尝试赋值无效，则可能会发生错误。有关有效性的详细信息，请参阅赋值。

如果源或目标元素具有内部结构，则 **move** 语句会将该结构视作类型为 CHAR 的字段，除非该结构的顶层指定了另一基本类型。在使用 **for count** 时，**move** 语句不考虑子结构。

当源和目标都是数组时，其中一个数组可能较长，当下面两个事件中的任何一个事件发生之后，操作就会结束：

- 在请求执行该操作的最后元素之间复制了数据；或者
- 在另一数组中具有匹配元素的最后一个元素中的数据复制完毕。

如果源不是数组，则下面两个事件中的任何一个事件发生之后，操作就会结束：

- 数据被复制至请求执行该操作的最后一个元素；或者
- 数据被复制至数组中的最后一个元素。

如果源是记录数组（或数组元素），则目标必须是记录数组。如果源是基本变量数组（或数组元素），则目标必须是基本变量数组或结构字段数组。如果源是结构字段数组（或数组元素），则目标必须是基本变量数组或结构字段数组。

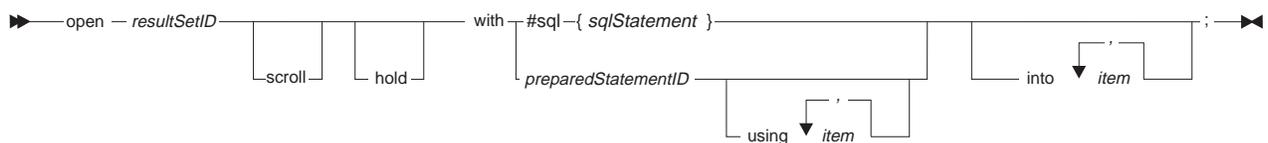
相关参考

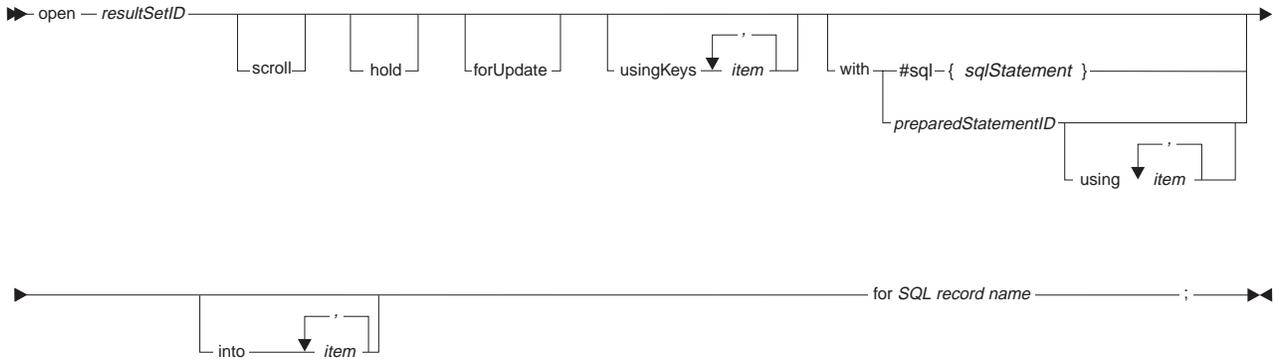
第 68 页的『数组』

第 340 页的『赋值』

open

EGL **open** 语句从关系数据库中选择一组行，以供 **get next** 语句以后检索。**open** 语句可以对游标或对被调用过程执行操作。





resultSetID

一个标识，它使 `open` 语句与以后的 `get next`、`replace`、`delete` 和 `close` 语句相关。有关详细信息，请参阅 *resultSetID*。

scroll

允许您在结果集中以各种方式移动的选项。语句 `get next` 总是可用的，但使用 `scroll` 还允许您使用下列语句：

- **get absolute**
- **get current**
- **get first**
- **get last**
- **get previous**
- **get relative**

hold

发生落实时保留结果集中的位置。

注：仅当 JDBC 驱动程序支持 JDBC 3.0 或更高版本时，`hold` 选项才可用。

hold 选项在下列情况下适用：

- 正在使用 EGL `open` 语句来打开游标而不是存储过程；并且
- 您想要定期落实更改而不丢失您在结果集中所处的位置；并且
- 数据库管理系统支持在 SQL 游标声明中使用 `WITH HOLD` 选项。

例如，代码可以执行下列操作：

1. 通过运行 EGL `open` 语句来声明并打开游标
2. 通过运行 EGL `get next` 语句来访问行
3. 在一个循环中执行以下操作：
 - a. 以某种方式处理数据
 - b. 通过运行 EGL `replace` 语句来更新行
 - c. 通过运行系统函数 `sysLib.commit` 来落实更改
 - d. 通过运行 EGL `get next` 语句来访问另一行

如果未指定 `hold`，则由于游标不再是打开的，所以步骤 3d 的第一次运行将失败。

在落实时不会关闭对其指定了 `hold` 的游标，但回滚或数据库连接将关闭所有游标。

如果不需要跨落实保留游标位置，则不要指定 `hold`。

forUpdate

一个选项，它允许以后使用 EGL 语句来替换或删除从数据库中检索到的数据。

如果正在调用存储过程以检索结果集，则不能指定 `forUpdate`。

usingKeys ... item

标识用来构建隐式 SQL 语句中的 WHERE 子句的键值部分的键项列表。如果未指定显式 SQL 语句，则在运行时将使用隐式 SQL 语句。

如果未指定 **usingKeys** 子句，则隐式语句的键值部分基于在 **open** 语句中引用的 SQL 记录部件。

如果指定显式 SQL 语句，则将忽略 **usingKeys** 信息。

with #sql{ sqlStatement }

显式 SQL SELECT 语句，如果还指定了 SQL 记录，则此语句是可选的。不要在 `#sql` 与左花括号之间留下任何空格。

into ... item

INTO 子句，它标识从游标或存储过程中接收值的 EGL 主变量。在与此子句类似的子句中（该子句位于 `#sql{ }` 块外部），不要在主变量名称之前包括分号。

with preparedStatementID

在运行时准备 SQL SELECT 或 CALL 语句的 EGL **prepare** 语句的标识。**open** 语句动态地运行 SQL SELECT 或 CALL 语句。有关详细信息，请参阅 *prepare*。

using ... item

USING 子句，它标识在运行时可供已准备的 SQL SELECT 或 CALL 语句使用的 EGL 主变量。在与此子句类似的子句中（该子句位于 `#sql{ }` 块外部），不要在主变量名称之前包括分号。

SQL record name

SQLRecord 类型的记录的名称。记录名或 *sqlStatement* 值是必需的；如果省略 *sqlStatement*，则 SQL SELECT 语句是从 SQL 记录派生的。

示例如下所示（采用名为 *emp* 的 SQL 记录）：

```
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)
}
```

缺省处理

当指定 SQL 记录时，缺省情况下，open 语句的作用如下所示：

- open 语句使一组行可用。所选行中的每一列都与一个结构项相关联，除了与只读结构项相关联的列以外，所有列都可以由 EGL replace 语句进行后续更新。
- 如果对 SQL 记录只声明一个键项，则只要 SQL 表键列中的值大于等于 SQL 记录的键项中的值，open 语句就会选择所有满足特定于记录的缺省选择条件的行。
- 如果对 SQL 记录声明了多个键，则特定于记录的缺省选择条件是唯一的搜索条件，并且 open 语句检索所有满足该条件的行。
- 如果即未指定记录键也未指定缺省选择条件，则 open 语句选择表中的所有行。
- 选择的行是未排序的。

在生成的代码中，EGL open 语句是由一个游标声明表示的，该游标声明包含 SQL SELECT 或 SQL SELECT FOR UPDATE 语句。缺省情况下，存在下列情况：

- FOR UPDATE 子句（如果有的话）不包含只读结构项
- 特定记录的 SQL SELECT 语句与以下语句类似：

```
SELECT column01,  
       column02, ...  
       columnNN  
INTO  :recordItem01,  
      :recordItem02, ...  
      :recordItemNN  
FROM  tableName  
WHERE keyColumn01 = :keyItem01  
FOR UPDATE OF  
      column01,  
      column02, ...  
      columnNN
```

可以通过在 EGL open 语句中指定 SQL 语句来覆盖缺省值。

错误状态

存在着各种各样的无效情况，包括：

- 包括了缺少 SELECT 的必需子句的 SQL 语句；必需子句包括 SELECT、FROM 和（如果指定 forUpdate 的话）FOR UPDATE OF
- SQL 记录与一列相关联，但该列在运行时不存在或者与相关结构项不兼容
- 指定了选项 forUpdate，并且代码尝试对下列任何一个 SQL 记录运行 open 语句：
 - 只包含只读结构项的 SQL 记录；或者
 - 与多个 SQL 表相关的 SQL 记录。

在下列情况下也可能发生问题：

1. 定制了 EGL open 语句以进行更新，但未指示特定 SQL 表列可用于更新；并且
2. 与 open 语句相关的 replace 语句尝试修改该列。

可以用下列任何一种方法来解决此问题：

- 定制 EGL open 语句时，在 SQL SELECT 语句的 FOR UPDATE OF 子句中包括列名；或者
- 定制 EGL replace 语句时，在 SQL UPDATE 语句的 SET 子句中除去对该列的引用；或者

- 对 **open** 和 **replace** 语句接受缺省值。

相关概念

- 第 125 页的『记录类型和属性』
- 第 209 页的『SQL 支持』
- 第 678 页的『resultSetID』
- 第 20 页的『对部件的引用』

相关任务

- 第 690 页的『EGL 语句和命令的语法图』

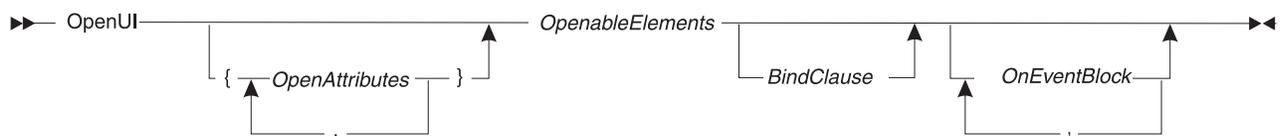
相关参考

- 第 511 页的『add』
- 第 517 页的『close』
- 第 520 页的『delete』
- 第 80 页的『EGL 语句』
- 第 86 页的『异常处理』
- 第 522 页的『execute』
- 第 533 页的『get』
- 第 544 页的『get next』
- 第 549 页的『get previous』
- 第 490 页的『I/O 错误值』
- 第 574 页的『prepare』
- 第 576 页的『replace』
- 第 61 页的『SQL 项属性』
- 第 860 页的『terminalID』

openUI

OpenUI 语句允许用户与界面基于 `consoleUI` 的程序交互。该语句定义用户和程序事件并指定如何响应每个事件。

OpenUI 语句的语法如下所示:



OpenAttributes

OpenAttributes 定义一组属性 - 值对，用逗号将每一对隔开，如以下示例中所示:

```
allowAppend = yes, allowDelete = no
```

每个属性都会影响用户交互，并且在某些情况下会覆盖在 *OpenableElements* 中引用的 `consoleUI` 变量的属性。属性如下所示:

allowAppend

指定用户是否能在屏幕上的 `arrayDictionary` 的末尾插入数据，如果设置为 `yes`，则有如下含义：

- 用户通过将光标移至 `arrayDictionary` 行来插入一行数据，该 `arrayDictionary` 行跟在包括数据的最后一行之后
- 该用户操作会将与该 `arrayDictionary` 绑定的元素追加至动态数组

用于变量类型: `ArrayDictionary`

属性类型: `Boolean`

示例: `allowAppend = no`

缺省值: `yes`；但在 `openUI` 属性 **`displayOnly`** 设置为 `yes` 时，缺省值为 `no`

allowDelete

指定用户是否能从屏幕上的 `arrayDictionary` 删除一行；如果设置为 `yes`，则有如下含义：

- 用户通过将光标移至某行并按 **`ConsoleLib.key_delete`** 中引用的键来删除该行。
- 该用户操作会删除与该 `arrayDictionary` 绑定的动态数组中的相关元素。

用于变量类型: `ArrayDictionary`

属性类型: `Boolean`

示例: `allowDelete = no`

缺省值: `yes`；但在 `openUI` 属性 **`displayOnly`** 设置为 `yes` 时，缺省值为 `no`

allowInsert

指定用户是否能在屏幕上的 `arrayDictionary` 中插入一行；如果设置为 `yes`，则有如下含义：

- 用户通过将光标移至现有行并按 **`ConsoleLib.key_insert`** 中引用的键来插入该行。
- 新行在光标所在的行之前
- 该用户操作会将与该 `arrayDictionary` 绑定的元素插入至动态数组

用于变量类型: `ArrayDictionary`

属性类型: `Boolean`

示例: `allowInsert = no`

缺省值: `yes`；但在 `openUI` 属性 **`displayOnly`** 设置为 `yes` 时，缺省值为 `no`

bindingByName

指示如何将一系列变量与一系列 `ConsoleField` 绑定；明确地说，是否将每个变量名与 `ConsoleField` 名相匹配。变量名列示在 `BindClause` 中，而 `ConsoleField` 名是 `ConsoleField` 名称字段中的值。

用于变量类型: `ConsoleForm`、`ConsoleField` 或 `Dictionary`；而不是 `arrayDictionary`

属性类型: `Boolean`

示例: `bindByName = yes`

缺省值: `no`

值如下所示：

no (缺省值)

将变量与 `ConsoleField` 按位置相匹配:

- 列表中的每个变量的位置; 以及
- `consoleForm` 中的每个 `ConsoleField` 的位置。

不管 `consoleField` 是显式列示在 `openUI` 语句中还是列示在字典声明中, 它们的顺序都将定义 `consoleField` 的顺序以便按位置进行绑定。(它们的顺序还将定义用户输入的跳进顺序, 如 *ConsoleUI 部件和相关变量*中所述。)

yes

将变量与 `ConsoleField` 按名称相匹配。

如果 `consoleField` 已列示或者在字典声明中(当绑定列表中没有匹配的变量时), `consoleField` 的用户输入会被忽略。同样, 没有与任何字段匹配的绑定变量也会被忽略。

在运行时, 至少一个 `consoleField` 和变量必须绑定在一起; 否则将发生错误。

color

指定 `ConsoleField` 中的文本的颜色。该值覆盖在 `ConsoleField` 声明中指定的颜色。

用于变量类型: *ConsoleForm*、*ConsoleField*、*ArrayDictionary* 或 *Dictionary*

属性类型: *ColorKind*

示例: `color = red`

缺省值: `white`

值如下所示:

defaultColor 或 white (缺省值)

白色

black

黑色

blue

蓝色

cyan

青色

green

绿色

magenta

品红色

red

红色

yellow

黄色

currentArrayCount

指定与屏幕上的 `arrayDictionary` 绑定的动态数组中的可用元素的数目。如果未指定此值, 则表示所有元素都可以在 `arrayDictionary` 中使用。

用于变量类型: *ArrayDictionary*

属性类型: *INT*

示例: *currentArrayCount = 4*

缺省值: *none*

displayOnly

指定 `consoleField` 是否仅供显示。如果设置为 *yes*, 则用户不能修改该数据, 该数据是受保护的, 不能更新它们。

用于变量类型: *ArrayDictionary*、*Dictionary*、*ConsoleField* 和 *ConsoleForm*

属性类型: *Boolean*

示例: *displayOnly = yes*

缺省值: *no*

help

指定用户按 **ConsoleLib.key_help** 中标识的键时要显示的文本。

此帮助文本是用于 **openUI** 命令的。在某些情况下, 与该键相关联的文本更加特定于上下文。例如, 菜单中的每个选项可以有自己的帮助消息。

用于变量类型: *ConsoleForm*、*ConsoleField*、*ArrayDictionary* 或 *Dictionary*

基本类型: *String*

示例: *help = "Update the value"*

缺省值: *Empty string*

helpKey

指定用于搜索资源束的访问键, 该资源束包含在出现以下情况时将显示的文本:

- 光标在 *OpenableElements* 中标识的 `ConsoleUI` 变量 (如 `ConsoleForm`) 中; 并且
- 用户按了 **ConsoleLib.key_help** 中标识的键。

如果同时指定 **help** 和 **helpKey**, 则使用 **help**。

用于变量类型: *ConsoleForm*、*ConsoleField*、*ArrayDictionary* 或 *Dictionary*

属性类型: *String*

示例: *helpKey = "myKey"*

缺省值: *Empty string*

资源束是由系统变量 **ConsoleLib.messageResource** 标识的, 如 *messageResource* 中所述。

highlight

指定显示 `ConsoleField` 时要使用的特殊效果 (如果有的话)。该值覆盖在 `ConsoleField` 声明中指定的等效值。

用于变量类型: *ConsoleForm*、*ConsoleField*、*ArrayDictionary* 或 *Dictionary*

属性类型: *HighlightKind[]*

示例: *highlight = [reverse, underline]*

缺省值: *[noHighLight]*

值如下所示:

noHighlight (缺省值)

不会有特殊效果。使用此值将覆盖任何其它值。

blink

没有效果。

reverse

反转文本和背景色，这样的话（举例来说），如果显示器是黑底白字的，则背景变为白色的，而文本变为黑色的。

underline

在受影响区域下面加下划线。下划线的颜色就是文本的颜色，即使同时指定了值 **reverse** 也是如此。

intensity

指定显示字体的强度。

用于变量类型: *ConsoleField*、*ConsoleForm*、*ArrayDictionary* 或 *Dictionary*

属性类型: *IntensityKind[]*

示例: *intensity = [bold]*

缺省值: *[normalIntensity]*

值如下所示:

normalIntensity (缺省值)

不会有特殊效果。使用此值将覆盖任何其它值。

bold

使文本以粗体字显示。

dim

当输入字段被禁用时，适当地让文本以较低强度出现。

invisible

除去任何有关“ConsoleField 在表单上”的指示。

isConstruct

指定 **openUI** 语句是否用于创建选择标准以供在 SQL 语句（如 SELECT）中使用。

用于变量类型: *ConsoleField*、*ConsoleForm* 和 *Dictionary*

属性类型: *Boolean*

示例: *isConstruct = no*

缺省值: *yes*

值如下所示:

no (缺省值)

与以往一样，每个 *ConsoleField* 与变量绑定。

yes

openUI 语句必须与字符类型的单个变量绑定。该变量不提供 *ConsoleField* 的初始值，但会接收用户输入，已经定义了该用户输入的格式以供在 SQL WHERE 子句中使用。

maxArrayCount

指定与屏幕上的 *arrayDictionary* 绑定的动态数组中的最大行数。达到最大数目后，用户将无法再插入行。

用于变量类型: *ArrayDictionary*

属性类型: *INT*

示例: *maxArrayCount = 20*

缺省值: *none*

setInitial

指定在用户修改 `ConsoleField` 的初始值（如 `consoleForm` 声明中定义的那样）之前是否显示该值。（可通过设置 `ConsoleField` 的 **initialValue** 字段来指定初始值。）

用于变量类型: *ConsoleField*、*ConsoleForm*、*Dictionary* 和 *ArrayDictionary*

属性类型: *Boolean*

示例: *setInitial = yes*

缺省值: *no*

如果 **setInitial** 的值为 **no**，则一开始将访存并显示绑定变量的值。

OpenableElements

可对其运行 **openUI** 语句的 `ConsoleUI` 变量:

- `ConsoleForm`
- `consoleField` 或下列其中一项:
 - 一系列 `consoleField`，每个 `consoleField` 用逗号隔开
 - 一个字典，它是在 `consoleForm` 中声明的，并且引用该 `consoleForm` 中的一组 `consoleField`
 - 一个 `arrayDictionary`，它是在 `consoleForm` 中声明的，并且引用该 `consoleForm` 中的一组 `consoleField` 数组
- 菜单
- 提示
- 窗口

BindClause

与 `ConsoleUI` 变量绑定的基本变量、记录或数组的列表。绑定变量的特征取决于对其运行 **openUI** 语句的 `consoleUI` 变量的特征:

- 对于 `consoleField`，可以指定基本变量。
- 对于屏幕上的 `arrayDictionary`，可以指定记录数组，以 `arrayDictionary` 中的每一行为一个元素；并且如果 `arrayDictionary` 中的每行表示单个值，可指定基本变量数组。
- 对于一个字典或一系列 `consoleField`，可指定一系列基本变量。或者可以指定记录数组，每个元素包含与 `consoleField` 绑定的一系列字段。此替代方法相当于将动态数组与屏幕上的 `arrayDictionary`（只有一行）绑定；可以追加、插入或删除记录以更改动态数组，在任何情况下，一次只能显示一个记录。
- 对于提示，可以指定接收用户响应的基本字段。

有关绑定的详细信息，请参阅有关 *OnEventBlock*（后面会讲到）以及 *ConsoleUI* 部件和相关变量的部分。

OnEventBlock

事件块是一种编程结构，它包括零到多个事件处理程序，每个事件处理程序包含为响应特定事件而编写的代码。下面是一个以 `OnEvent` 头开始的事件处理程序:

```
OnEvent(eventKind: eventQualifiers)
```

eventKind

其中一个事件。有效值在『事件类型』中作了描述。

eventQualifier

进一步定义事件的数据。这种数据可能是输入的 `ConsoleField` 或按下的击键。

响应指定事件的 EGL 语句在一个 `OnEvent` 头与下一个 `OnEvent` 头（如果有的话）之间，如后面的示例中所示。

用户继续与程序交互，并且在发生与事件处理程序相关联的事件时，该程序将运行该事件处理程序。如果 `openUI` 语句用于显示提示，则用户程序交互不会循环：

1. 事件处理程序（可能是其中一个）捕获用户击键并作出响应
2. `openUI` 语句结束

窗口没有可用的事件块。

考虑以下示例来了解用户与 `ConsoleForm` 之间的交互：

```
openUI {bindingByName=yes}
  activeForm
  bind firstName, lastName, ID
  OnEvent(AFTER_FIELD:"ID")
    if (employeeID == 700)
      firstName = "Angela";
      lastName = "Smith";
    end
end
```

该代码执行下列操作：

- 打开活动的 `ConsoleForm`（即最新显示在活动窗口中的 `consoleForm`）；
- 将一组基本变量与每个 `ConsoleField` 绑定；并且
- 指定用户在 `employeeID` 中输入值并离开该 `ConsoleField` 之后，EGL 会将字符串放在其它两个变量中。

考虑有关上述示例的以下详细信息：

- 光标从列出的 `consoleField` 中的第一项中开始；但应该从标识 `consoleField` 中开始以便用户在其其它 `consoleField` 中的输入不会被事件处理程序擦除。
- 事件处理程序更新与 `firstName` 和 `lastName` `consoleField` 绑定的变量，并且在光标进入这些字段之前不会显示这些值。您以前可能想要显示这些值。

可通过以 `exit openUI` 的格式发出 `exit` 语句来结束 `openUI` 语句。

事件类型

`ConsoleUI` 支持下列事件：

BEFORE_OPENUI

EGL 运行时开始运行 `OpenUI` 语句。此事件可供基于窗口的变量之外的所有 `ConsoleUI` 变量使用。

AFTER_OPENUI

EGL 运行时将停止运行 `OpenUI` 语句。此事件可供基于窗口的变量之外的所有 `ConsoleUI` 变量使用。

ON_KEY:(ListOfStrings)

用户按了特定键，如“ESC”、“F2”或“CONTROL_W”之类的字符串所示。可通过隔开每个字符串来标识多个键，如以下示例中所示：

```
ON_KEY:("a", "ESC")
```

此事件可供基于窗口的变量之外的所有 ConsoleUI 变量使用。

BEFORE_FIELD:(ListOfStrings)

用户已将光标移到指定的 ConsoleField 中，它是由与 ConsoleField 名称字段的值匹配的字符串表示的。可通过隔开每个字符串来标识同一 consoleForm 中的多个 ConsoleField，如以下示例中所示：

```
BEFORE_FIELD:("field01", "field02")
```

AFTER_FIELD:(ListOfStrings)

用户已将光标移出指定的 ConsoleField，就像与 ConsoleField 名称字段的值匹配的字符串指示的那样。可通过隔开每个字符串来标识同一 consoleForm 中的多个 ConsoleField，如以下示例中所示：

```
AFTER_FIELD:("field01", "field02")
```

BEFORE_DELETE

对于屏幕上的 arrayDictionary，用户按了 **ConsoleLib.key_deleteLine** 中指定的键，但 EGL 运行时还未删除行。该程序可以调用 **consoleLib.cancelDelete** 以避免删除该行。

BEFORE_INSERT

对于屏幕上的 arrayDictionary，用户按了 **ConsoleLib.key_insertLine** 中指定的键，但 EGL 运行时还未插入行。该程序可以调用 **consoleLib.cancelInsert** 以避免插入该行。

BEFORE_ROW

对于屏幕上的 arrayDictionary，用户将光标移到行中。

AFTER_DELETE

对于屏幕上的 arrayDictionary，用户按了 **ConsoleLib.key_deleteLine** 中指定的键，并且 EGL 运行时已删除行。

AFTER_INSERT

对于屏幕上的 arrayDictionary，用户按了 **ConsoleLib.key_insertLine** 中指定的键；EGL 运行时已插入行；并且光标离开已插入的行。

在将更改落实至数据库之前，可以编辑该行；就像 AFTER_INSERT 处理程序中经常发生的那样。

AFTER_ROW

用户从屏幕上的 arrayDictionary 中的行移动光标。

MENU_ACTION:(ListOfStrings)

用户选择了 menuItem，就像与 menuItem 名称字段的值相匹配的字符串所指示的那样。可通过隔开每个字符串来标识多个 menuItem，如以下示例中所示：

```
MENU_ACTION:("item01", "item02")
```

isConstruct

当属性 **isConstruct** 为 *yes* 时，放在与 **openUI** 命令绑定的变量中的文本的格式是以特殊方式定义的，如以下示例中所示：

1. openUI 语句将针对由 3 个 ConsoleField (*employee*、*age* 和 *city*) 组成的 ConsoleForm 运行，并且每个字段与同名的 SQL 表列相关联。

通过设置 consoleField 属性 **SQLColumnName** 将 consoleField 与 SQL 表列相关联；必须设置 consoleField 属性 **dataType**，如 *ConsoleField* 属性和字段中所示。

2. 用户执行下列操作：

- 将 *employee* 字段留为空白
- 在 *age* 字段中输入以下内容：
 > 25
- 在 *city* 字段中输入以下内容：
 = 'Sarasota'

3. 当用户离开屏幕上对其运行 openUI 语句的变量时，绑定变量接收以下内容：

AGE > 28 AND CITY = 'Sarasota'

就像显示的那样，EGL 将运算符 AND 放在用户提供的每个子句之间。

下表显示有效的用户输入和生成的子句。短语简单 SQL 类型指的是未结构化也非类似 LOB 类型的 SQL 类型。

符号	定义	受支持的数据类型	示例	生成的子句 (对于名为 C 的字符列)	生成的子句 (对于名为 C 的数字列)
=	等于	简单 SQL 类型	=x, ==x	C = 'x'	C = x
>	大于	简单 SQL 类型	>x	C > 'x'	C > x
<	小于	简单 SQL 类型	<x	C < 'x'	C < x
>=	不小于	简单 SQL 类型	>=x	C >= 'x'	C >= x
<=	不大于	简单 SQL 类型	<=x	C <= 'x'	C <= x
<> 或 !=	不等于	简单 SQL 类型	<>x 或 !=x	C != 'x'	C != x
..	范围	简单 SQL 类型	x.y 或 x..y	C BETWEEN 'x' AND 'y'	C BETWEEN x AND y
*	字符串的通配符 (如下一个表中所述)	CHAR	*x 或 x* 或 *x*	C MATCHES '*x'	不适用
?	单字符通配符 (如下一个表中所述)	CHAR	?x, x?, ?x?, x??	C MATCHES '?x'	不适用
	逻辑或	简单 SQL 类型	x y	C IN ('x', 'y')	C IN (x, y)

等号 (=) 可以表示 IS NULL；而不等号 (!= 或 <>) 可以表示 IS NOT NULL。

- 在运行 **prepare** 语句之后，可以针对 I/O 错误值来测试记录名，以确定该语句是否成功，如以下示例所示：

```

try
  prepare prep01 from
    "insert into " + aTableName +
    "(empnum, empname) " +
    "value ?, ?"
  for empRecord;

onException
  if empRecord is unique
    myErrorHandler(8);
  else
    myErrorHandler(12);
  end
end
end

```

下面是另一个示例：

```

myString =
  "insert into myTable " + "(empnum, empname) " +
  "value ?, ?";

try
  prepare myStatement
  from myString;
onException
  myErrorHandler(12);    // exits the program
end

try
  execute myStatement
  using :myRecord.empnum,
        :myRecord.empname;
onException
  myErrorHandler(15);
end

```

如上述示例所示，开发者可以在应该出现主变量的位置使用问号（?）。于是，在运行时使用的主变量的名称被放在运行已编译语句的 **execute**、**open** 或 **get** 语句的 **using** 子句中。

对结果集的一行执行操作的 **prepare** 语句可以包含一个具有 *where current of resultSetIdentifier* 格式的短语。此项技术只有在下列情况下才有效：

- 在文字中编写该短语；并且
- 当运行 **prepare** 语句时，结果集已打开。

下面是一个示例：

```

prepare prep02 from
  "update myTable " +
  "set empname = ?, empphone = ? where current of x1" ;

execute prep02 using empname, empphone ; freeSQL prep02;

```

有关圆括号对加号（+）用法的影响的示例，请参阅表达式。

相关概念

第 20 页的『对部件的引用』

第 125 页的『记录类型和属性』

第 209 页的『SQL 支持』

相关参考

第 511 页的『 add 』
第 517 页的『 close 』
第 520 页的『 delete 』
第 86 页的『 异常处理 』
第 522 页的『 execute 』
第 452 页的『 表达式 』
第 532 页的『 freeSQL 』
第 533 页的『 get 』
第 544 页的『 get next 』
第 549 页的『 get previous 』
第 490 页的『 I/O 错误值 』
第 80 页的『 EGL 语句 』
第 561 页的『 open 』
『 replace 』
第 61 页的『 SQL 项属性 』
第 462 页的『 文本表达式 』
第 690 页的『 EGL 语句和命令的语法图 』

print

EGL **print** 语句将一个打印表单添加至运行时缓冲区，如打印表单中所述。

```
▶ print printFormName ;
```

printFormName

对程序可视的打印表单的名称。有关可视性的详细信息，请参阅对部件的引用。

相关概念

第 144 页的『 打印表单 』
第 20 页的『 对部件的引用 』

replace

EGL **replace** 语句将已更改的记录放到文件或数据库中。

```
▶▶ replace ————— ;
```

<i>indexed record name</i>	—————
<i>relative record name</i>	—————
SQL <i>record name</i>	—————
with #sql{ <i>sqlStatement</i> }	from <i>resultSetID</i>

record name

I/O 对象的名称：带索引记录、相对记录或 SQL 记录。

with #sql{ *sqlStatement* }

显式 SQL UPDATE 语句。不要在 #sql 与左花括号之间留下任何空格。

from *resultSetID*

一个标识，它使 **replace** 语句与同一程序中之前运行的 **get** 或 **open** 语句相关。有关详细信息，请参阅 *resultSetID*。

以下示例显示如何读取和替换文件记录：

```
emp.empnum = 1;           // sets the key in record emp

try
  get emp forUpdate;
onException
  myErrorHandler(8); // exits the program
end

emp.empname = emp.empname + " Smith";

try
  replace emp;
onException
  myErrorHandler(12);
end
```

有关 **replace** 语句的详细信息取决于记录类型。有关 SQL 处理的详细信息，请参阅 *SQL 记录*。

带索引记录或相对记录

如果要替换带索引记录或相对记录，则必须对该记录发出带有 **forUpdate** 选项的 **get** 语句，然后发出 **replace** 语句，在此之间不对同一个文件执行任何 I/O 操作。在调用 **replace** 语句之后，对同一文件执行的下一个 I/O 操作的作用如下所示：

- 如果下一个 I/O 操作是对同一个 EGL 记录执行的 **replace** 语句，则在文件中更改该记录
- 如果下一个 I/O 操作是对同一个 EGL 记录执行的 **delete** 语句，则在文件中对该记录设置删除标记
- 如果下一个 I/O 操作是对同一文件中的记录执行的 **get** 语句，并包含 **forUpdate** 选项，则后续 **replace** 或 **delete** 语句对新读取的文件记录有效
- 如果下一个 I/O 操作是对同一个 EGL 记录执行的 **get** 语句（不带 **forUpdate** 选项）或者是对同一个文件执行的 **close** 语句，则释放文件记录而不进行更改

有关 **forUpdate** 选项的详细信息，请参阅 *get*。

SQL 记录

对于 SQL 处理，EGL **replace** 语句致使生成的代码包含 SQL UPDATE 语句。

必须通过两种方法中的任何一种来检索一行以进行后续替换：

- 发出 **get** 语句（带有 **forUpdate** 选项）以检索该行；或者
- 发出 **open** 语句来选择一组行，然后调用 **get next** 语句来检索所关心的行。

错误状态： 当使用 **replace** 语句时，无效的情况包括：

- 指定了不是 UPDATE 类型的 SQL 语句
- 指定了 SQL UPDATE 语句的一些子句但不是所有子句
- 当 *resultSetID* 值是必需的时未指定该值；有关详细信息，请参阅 *resultSetID*

- 指定了（或接受了）具有下列其中一个特征的 UPDATE 语句：
 - 更新多个表
 - 与不存在或者与相关主变量不兼容的列相关联
- 使用了 SQL 记录作为 I/O 对象，并且所有记录项都是只读的

下列情况也会导致错误：

- 定制了带有 forUpdate 选项的 EGL **get** 语句，但未指示特定 SQL 表列可用于更新；并且
- 与 **get** 语句相关的 **replace** 语句尝试修改该列。

可以通过下列任何一种方法来解决前述不匹配问题：

- 定制 EGL **get** 语句时，在 SQL SELECT 语句的 FOR UPDATE OF 子句中包括列名；或者
- 定制 EGL **replace** 语句时，在 SQL UPDATE 语句的 SET 子句中除去对该列的引用；或者
- 对 **get** 和 **replace** 语句接受缺省值。

隐式 SQL 语句： 缺省情况下，写 SQL 记录的 **replace** 语句的作用如下所示：

- 作为记录声明中的记录项与 SQL 表列相关联的结果，生成的代码将每个记录项中的数据复制到相关 SQL 表列中
- 如果已将记录项定义为只读的，则与该记录项相对应的列中的值不受影响

缺省情况下，SQL 语句具有下列特征：

- SQL UPDATE 语句不包含只读的记录项
- 特定记录的 SQL UPDATE 语句类似于以下语句：

```
UPDATE tableName
SET column01 = :recordItem01,
    column02 = :recordItem02,
    .
    .
    .
    columnNN = :recordItemNN WHERE CURRENT OF cursor
```

相关概念

- 第 125 页的『记录类型和属性』
- 第 20 页的『对部件的引用』
- 第 678 页的『resultSetID』
- 第 678 页的『运行单元』
- 第 209 页的『SQL 支持』

相关任务

- 第 690 页的『EGL 语句和命令的语法图』

相关参考

- 第 511 页的『add』
- 第 517 页的『close』
- 第 520 页的『delete』
- 第 80 页的『EGL 语句』
- 第 86 页的『异常处理』

第 522 页的『execute』
第 533 页的『get』
第 544 页的『get next』
第 549 页的『get previous』
第 490 页的『I/O 错误值』
第 561 页的『open』
第 574 页的『prepare』
第 61 页的『SQL 项属性』
第 860 页的『terminalID』

return

EGL **return** 语句从函数中退出并返回（可选）一个值给调用函数。



returnValue

与 EGL 函数声明中的 **returns** 规范相兼容的项、文字或常量。

虽然一个项在所有方面都必须与 **returns** 规范相对应，但是文字和常量的规则如下所示：

- 仅当 **returns** 规范中的基本类型是数字类型时才可以返回数字文字或常量
- 仅当 **returns** 规范中的基本类型是 CHAR 或 MBCHAR 时才可以返回只包含单字节字符的文字或常量
- 仅当 **returns** 规范中的基本类型是 DBCHAR 时才可以返回只包含双字节字符的文字或常量
- 仅当 **returns** 规范中的基本类型是 MBCHAR 时才可以返回包含单字节字符与双字节字符的组合的文字或常量
- 如果 **returns** 规范中的基本类型是 HEX，则不能返回文字或常量

包含 **returns** 规范的函数必须以包含值的 **return** 语句终止。不包含 **returns** 规范的函数可以以一定不能包含值的 **return** 语句终止。

return 语句将控制权传递给函数调用之后的第一个语句，即使该语句位于 **try** 块中的 **OnException** 子句中。

set

下列各节描述 EGL **set** 语句的作用：

- 第 580 页的『对整个记录（或固定记录）的影响』
- 第 581 页的『对整个表单的作用』
- 第 582 页的『对任何上下文中的字段的影响』
- 第 583 页的『对文本表单中的字段的作用』

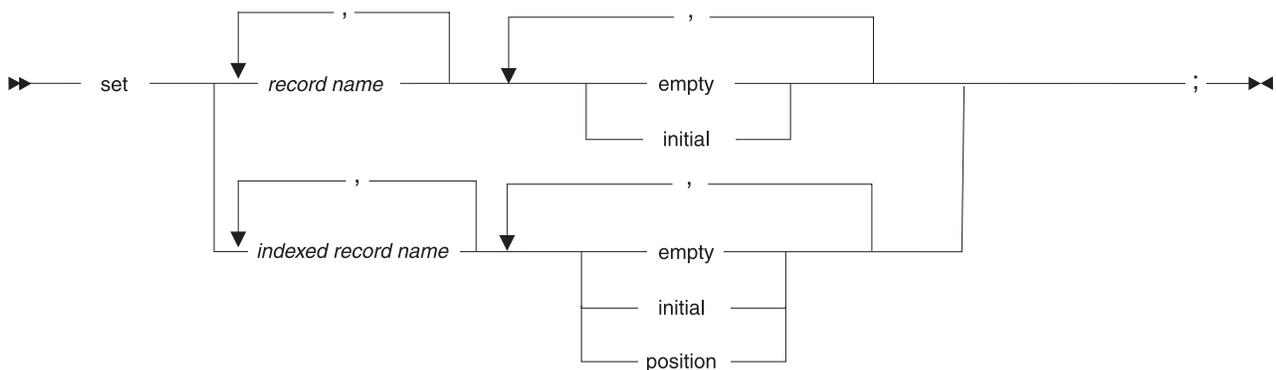
对整个记录（或固定记录）的影响

下表描述影响整个记录或固定记录或者记录或固定记录数组的 **set** 语句。

set 语句的格式	作用
set record empty	<p>清空每个基本字段。对于记录，将清空每个下级记录，清空这些下级记录的每个下级记录，以此类推。对于固定记录（它本身可能在记录中），基本字段位于固定结构的最低级别。</p> <p>每个基本字段的影响取决于该字段的基本类型：</p> <ul style="list-style-type: none"> 对于类型为 ANY 的字段，set 语句会根据字段的当前类型对字段进行初始化；并且如果字段为 ANY 类型并且不具有任何其它类型，则 set 语句将不起作用 有关其它类型的字段的详细信息，请参阅数据初始化。
set record initial	<p>将字段值复位为由 value 属性在开发时指定的值，这对于在 pageHandler 或表单中声明的记录或固定记录是可行的。由赋值设置的值永远不会复位。</p> <p>如果 value 属性没有值或者如果记录不在 pageHandler 或表单中，<i>set record initial</i> 的效果与 <i>set record empty</i> 的效果相同，但以下情况例外：对于类型为 ANY 的字段，set 语句将除去 ANY 之外的所有类型说明。</p>
set record position	<p>确定与类型为 indexedRecord 的固定记录相关联的 VSAM 文件中的位置，如稍后所述。</p> <p>此 set 语句格式对数组不可用。</p>

可将各种语句格式组合使用，选项间用逗号隔开。对于给定记录，选项按它们在 **set** 语句中的出现顺序起作用。您还可以指定多个记录，记录间用逗号隔开。

语法图如下所示：



record name

任何类型的记录或固定记录的名称。可以指定数组。

indexed record name

类型为 `indexedRecord` 的固定记录的名称。仅当未包括 *set record position* 时，才能指定数组。

empty

如上表所述。

initial

如上表所述。

position

根据 *set value*（这是带索引记录中的键值）确定文件位置。整体作用取决于代码对同一个带索引记录执行的下一个输入或输出操作：

- 如果下一个操作是 EGL **get next** 语句，则该语句读取键值等于或大于 *set value* 的第一个文件记录。如果不存在这样的记录，则 **get next** 语句的结果是 **endOfFile**。
- 如果 *set record position* 之后的下一个操作是 EGL **get previous** 语句，则该语句读取键值等于或小于 *set value* 的第一个文件记录。如果不存在这样的记录，则 **get previous** 的结果是 **endOfFile**。
- *set record position* 之后的任何其它操作都将重置文件位置，并且 *set record position* 不起作用。

如果用十六进制 FF 值填充了 *set value*，则下列情况成立：

- *set record position* 在文件中的最后一条记录之后确定文件位置
- 如果下一个操作是 **get previous** 语句，则检索文件中的最后一条记录

对整个表单的作用

下表描述影响整个表单的 **set** 语句。

set 语句的格式	作用
set form alarm	仅限于文本表单；当下一次 converse 语句显示表单时发出警报声。
set form empty	清空表单中的每个字段的值，清除任何内容。对给定字段的影响取决于基本类型： <ul style="list-style-type: none"> • 对于类型为 ANY 的字段，set 语句会根据字段的当前类型对字段进行初始化；并且如果字段为 ANY 类型并且不具有任何其它类型，则 set 语句将不起作用 • 有关其它类型的字段的详细信息，请参阅数据初始化。
set form initial	将每个表单字段重置为其最初定义的状态，该状态是在表单声明中指定的。程序所作的更改都将被取消。对于类型为 ANY 的字段， set 语句会除去 ANY 之外的所有类型说明。

set 语句的格式	作用
set form initialAttributes	将每个表单字段重置为其最初定义的状态，该状态是在表单声明中指定的。字段的内容不受影响，类型也不受影响（如果字段类型为 ANY 的话）。

可以将各种语句格式组合使用，选项（如 **empty** 和 **alarm**）间用逗号隔开。您还可以指定多个表单，表单间用逗号隔开。

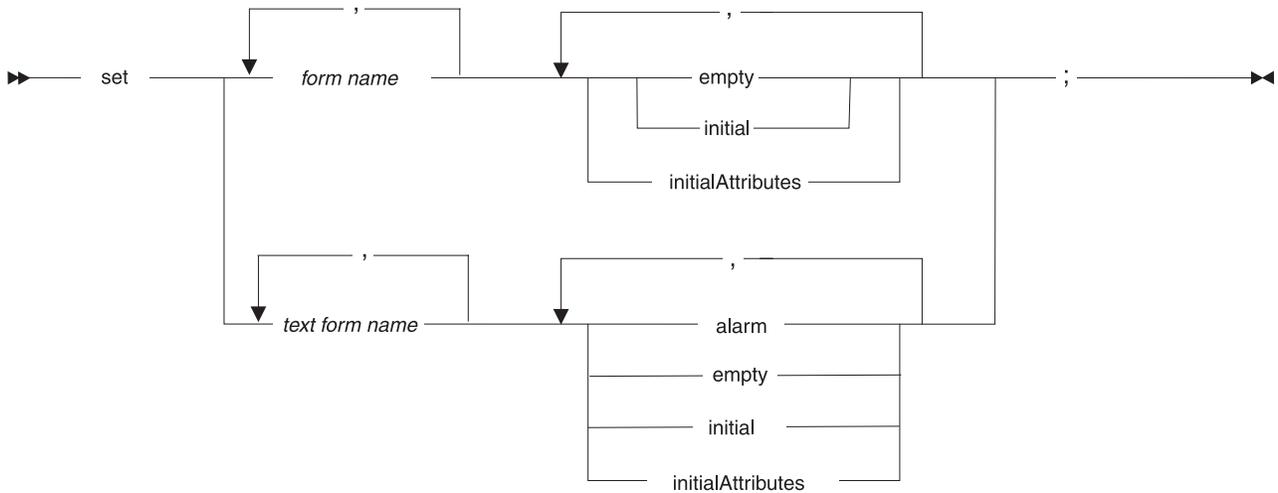
对于下列格式，可以选择一种格式，也可以不选择：

- *set form empty*
- *set form initial*

对于下列格式，可以选择一种格式、两种格式或不选择：

- *set form alarm*（仅可用于文本表单）
- *set form initialAttributes*

语法图如下所示：



form name

text 类型或 *print* 类型的表单的名称，如表单部件中所述。

text form name

text 类型的表单的名称，如表单部件中所述。

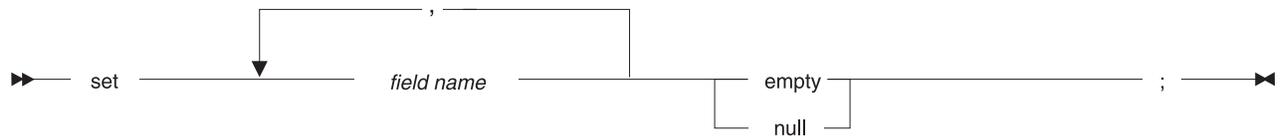
上表对这些选项作了描述。

对任何上下文中的字段的影响

下表描述影响任何上下文中的字段的 **set** 语句的格式。

set 语句的格式	作用
set field empty	清空字段或（对于具有子结构的固定字段）清空每个下级基本字段。 它的影响取决于字段的基本类型： <ul style="list-style-type: none"> 对于类型为 ANY 的字段，set 语句会根据字段的当前类型对字段进行初始化；并且如果字段为 ANY 类型并且不具有任何其它类型，则 set 语句将不起作用 有关其它类型的字段的详细信息，请参阅数据初始化。
set field null	使字段为 NULL（如果这样做有效的话）。有关此操作何时有效的详细信息，请参阅 <i>itemsNullable</i> 。有关 SQL 记录中的 NULL 处理的详细信息，请参阅 <i>SQL 项属性</i> 。

语法图如下所示：



field name

字段的名称。

您可以选择一个或另一个选项，每个选项在上表中作了描述。

对文本表单中的字段的作用

下表描述影响文本表单中的字段或字段数组的 set 语句。给定的 set 语句只能按一组特定的方式来组合选项，如稍后所述。

注：描述的许多操作都取决于显示文本表单的设备。建议您在所支持的每个设备上测试输出。

set 语句的格式	作用
set field blink	导致文本重复地闪烁。此选项仅在 COBOL 程序中可用。
set field bold	使文本以粗体字显示。
set field cursor	将光标定位在指定的字段中。 如果字段标识了数组，并且没有 occurs 值，则缺省情况下光标定位在第一个数组元素上。 如果程序运行多个具有 <i>set field cursor</i> 格式的语句，则最后一个语句在 converse 语句运行时起作用。

set 语句的格式	作用
set field defaultColor	将特定于字段的 color 属性设置为 <i>defaultColor</i> ，这表示显示的颜色由其它条件确定。有关详细信息，请参阅字段显示属性。
set field dim	导致字段以低于正常的强度显示。使用此效果来淡化字段内容。
set field empty	初始化字段的值，并清除任何内容。对给定字段的作用取决于基本类型，如数据初始化中所述。
set field full	<p>在显示表单之前，将空的、空白的或 NULL 字段设置为一系列完全相同的字符：</p> <ul style="list-style-type: none"> • 如果字段属性 fillCharacter 是以下值（这也是 fillCharacter 的缺省值），则该字符为星号 (*)： <ul style="list-style-type: none"> - 对于 HEX 类型的字段，值为 0 - 对于数字类型的字段，值为空格 - 对于其它字段，值为空字符串 • 如果未将 fillCharacter 设置为上面描述的值，则该字符与 fillCharacter 的值完全相同。 <p>仅当针对该字段的已修正数据标记被设置好时，才将表单上的字符返回给程序，如已修正数据标记和 <i>modified</i> 属性中所述。更改字段的用户必须除去字段中的所有字符才能避免将它们返回给程序。</p> <p>仅当表单组是使用构建描述符选项 <i>setFormItemFull</i> 生成时，<i>set field full</i> 的使用才起作用。</p> <p>如果 MBCHAR 类型的字段只包含单字节空格，则认为它是空的。对于这样的字段，<i>set field full</i> 指定一系列单字节字符。</p>
set field initial	将字段重置为其最初定义的状态，和程序所作的任何更改无关
set field initialAttributes	将字段重置为其最初定义的状态，而不使用 value 属性（该属性指定字段的当前内容）
set field invisible	使字段文本不可视
set field masked	适用于密码字段。如果文本表单是由 Java 程序提供的，将显示星号而不是用户在输入字段中输入的任何非空白字符。
set field modified	设置已修正数据标记，如已修正数据标记和 <i>modified</i> 属性中所述。
set field noHighlight	消除闪烁、反转和下划线等特殊效果。

set 语句的格式	作用
set field normal	重置字段，如下列格式所述： <ul style="list-style-type: none"> • Set field normalIntensity • Set field unmodified • Set field unprotected 有关详细信息，请参阅下表。
set field normalIntensity	将字段设置为可视，并且没有粗体字效果。
set field protect	设置字段，使用户不能覆盖其中的值。另请参阅 <i>set field skip</i> 。
set field reverse	反转文本和背景色，举例来说，如果显示器是黑底白字的，则背景变为白色的，而文本变为黑色的。
set field <i>selectedColor</i>	将特定于字段的 color 属性设置为指定的值。 <i>selectedColor</i> 的有效值如下所示： <ul style="list-style-type: none"> • black • blue • green • pink • red • turquoise • white • yellow
set field skip	设置字段，使用户不能覆盖其中的值。另外，在下列任何一种情况下，光标将跳过该字段： <ul style="list-style-type: none"> • 用户正在依据跳进顺序在上一个字段中输入，并且按下 Tab 键或者在上一个字段填充了内容；或者 • 用户正在依据跳进顺序在下一个字段中输入，并且按下 Shift Tab 键。
set field underline	在字段底部放置一条下划线。
set field unprotect	设置字段，使用户可以覆盖其中的值。

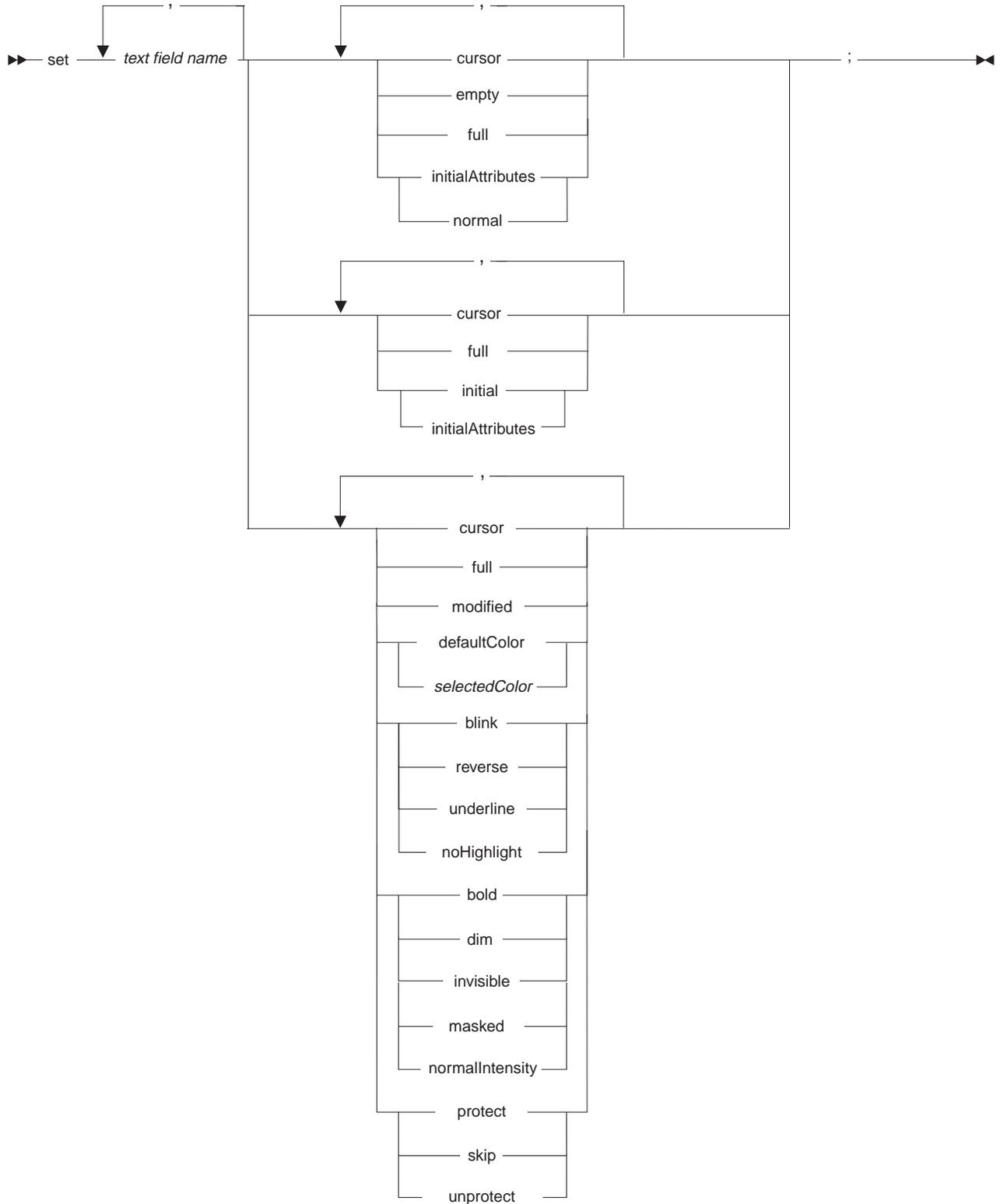
可以按照三种方式中的任何一种方式来将各种语句格式组合使用，并插入逗号以将选项（如 **cursor** 和 **full**）分隔开：

1. 可以按照以下方式构造 **set** 语句：

- 选择下列其中一种字段属性格式或不选择任何字段属性格式：
 - *set field initialAttributes*
 - *set field normal*
- 选择任意数目的下列格式：
 - *set field cursor*
 - *set field empty*
 - *set field full*

2. 其次，可以使用任意数目的下列格式来构造 **set** 语句：
 - *set field cursor*
 - *set field full*
 - *set field initial* 或 *set field initialAttributes*
3. 最后，可以按照以下方式构造 **set** 语句：
 - 选择任意数目的下列格式：
 - *set field cursor*
 - *set field full*
 - *set field modified*
 - 选择下列其中一种颜色格式或不选择任何颜色格式：
 - *set field defaultColor*
 - *set field selectedColor*
 - 选择下列其中一种突出显示格式或不选择任何突出显示格式：
 - *set field blink*
 - *set field reverse*
 - *set field underline*
 - *set field noHighlight*
 - 选择下列其中一种强度格式或不选择任何强度格式：
 - *set field bold*
 - *set field dim*
 - *set field invisible*
 - *set field masked*
 - *set field normalIntensity*
 - 选择下列其中一种保护格式或不选择任何保护格式：
 - *set field protect*
 - *set field skip*
 - *set field unprotect*

语法图如下所示：



field name

文本表单中的字段的名称。该名称可能指的是字段数组。

上表对这些选项作了描述。

相关概念

第 142 页的『表单部件』

第 148 页的『已修正数据标记和 `modified` 属性』

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 430 页的『数据初始化』

第 80 页的『EGL 语句』

第 60 页的『字段显示属性』

第 544 页的『`get next`』

第 549 页的『`get previous`』

第 359 页的『`itemsNullable`』

第 61 页的『SQL 项属性』

show

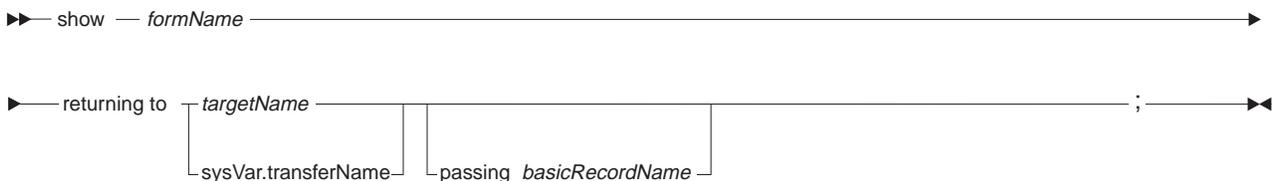
show 语句从主程序显示文本表单:

1. 落实可恢复的资源、关闭文件并释放锁
2. (可选) 传递基本记录以供 **show** 语句的 `returning` 子句 (如果有的话) 中指定的程序使用
3. 结束第一个程序
4. 显示文本表单

show 语句在被调用程序中不可用。

如果在 **show** 语句中包括了 `returning` 子句, 则当用户按下事件键时, EGL 运行时将调用指定的程序。将把表单数据赋予接收程序的输入表单。将把传递的记录 (不被用户输入更改) 赋予接收程序的输入记录。

如果未包括 `returning` 子句, 则当显示文本表单时, 操作结束。



formPartName

对程序可视的文本表单的名称。有关可视性的详细信息, 请参阅对部件的引用。如果在语句中包括了 `returning` 子句, 则文本表单必须与正被调用的程序的 `inputForm` 属性中指定的文本表单相同。

sysVar.transferName

一个系统变量, 它包含将要调用的程序的标识。使用此变量来在运行时设置标识。

basicRecordName

类型为 `basicRecord` 的记录的名称。将把内容赋予接收程序的输入记录。

相关概念

第 20 页的『对部件的引用』

相关参考

第 861 页的『transferName』

transfer

EGL **transfer** 语句将控制权从一个主程序转移到另一个主程序，结束转移程序，并传递（可选）一条记录，该记录的数据被接受到接收程序的输入记录中。不能在被调用程序中使用 **transfer** 语句。

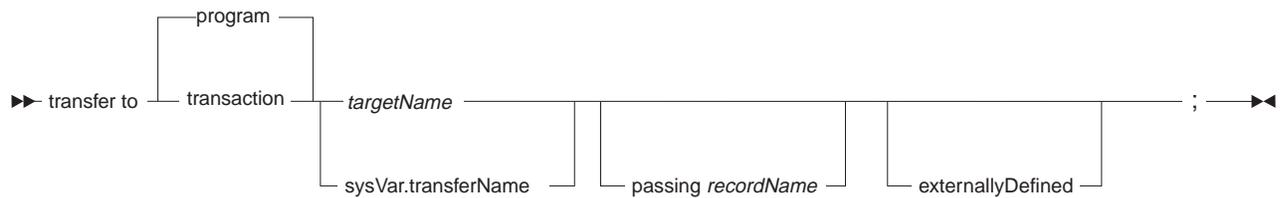
程序可以通过 *transfer to a transaction* 格式的语句来转移控制权：

- **transfer to a transaction** 执行下列操作：
 - 在作为 Java 主文本程序或主批处理程序运行的程序中，行为取决于构建描述符选项 **synchOnTrxTransfer** 的设置：
 - 如果 **synchOnTrxTransfer** 的值为 YES，则 **transfer** 语句将落实可恢复的资源、关闭文件、关闭游标并在同一个运行单元中启动程序。
 - 如果 **synchOnTrxTransfer** 的值为 NO（缺省值），则 **transfer** 语句也在同一个运行单元中启动程序，但不关闭或落实可供被调用程序使用的资源。
 - 在 PageHandler 中，**transfer to a transaction** 是无效的；请改为使用 **forward** 语句。

当正在将控制权从 Java 代码转移至 Java 代码时，链接选项部件的 **transferLink** 元素不起作用，但在别的情况下是有意义的。

如果正在将控制代码转移至不是使用 EGL 或 VisualAge Generator 编写的代码，则建议您设置链接选项部件的 **transferLink** 元素。将 **linkType** 属性设置为 *externallyDefined*。

如果正在以 VisualAge Generator 兼容性方式运行，则可以在 **transfer** 语句中指定选项 **externallyDefined**，对于从 VisualAge Generator 迁移的程序，就是这种情况；但建议您改为在链接选项部件中设置等效的值。有关 VisualAge Generator 兼容性方式的详细信息，请参阅与 *VisualAge Generator* 的兼容性。



program *targetName* (缺省值)

接收控制权的程序。

transaction *targetName*

接收控制权的程序，如之前所述。

sysVar.transferName

包含可以在运行时设置的目标名的系统函数。有关详细信息，请参阅 *sysVar.transferName*。

passing recordName

作为目标程序中的输入记录进行接收的记录。传递的记录可以是任何类型，但长度和基本类型必须与接收数据的记录兼容。目标程序中的输入记录必须具有 *basicRecord* 类型。

相关概念

第 400 页的『与 VisualAge Generator 的兼容性』

第 608 页的『为名称取别名』

相关参考

第 861 页的『transferName』

try

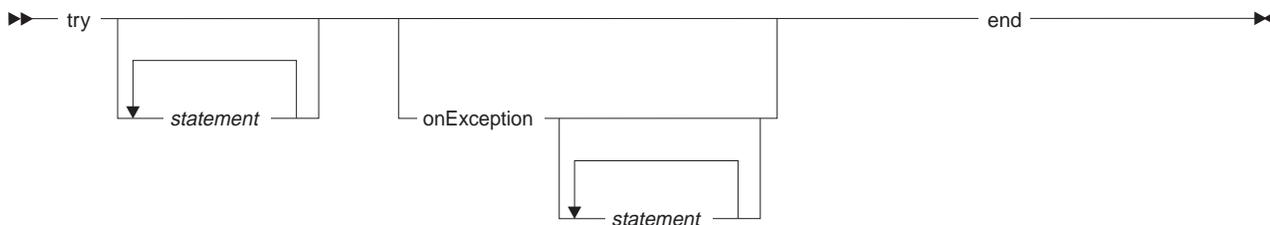
EGL **try** 语句指示当下列任何类型的语句导致错误并且该语句位于 **try** 语句中时，程序继续运行：

- 输入 / 输出 (I/O) 语句
- 系统函数调用
- **call** 语句

如果发生异常，则在 **onException** 块（如果有的话）中的第 1 个语句处继续处理，或者在 **try** 语句后面的第 1 个语句处继续处理。但是，仅当系统变量 **VGVar.handleHardIOErrors** 设置为 1 时才处理硬 I/O 错误；否则，程序显示消息（如果有可能的话）并结束。

当异常发生在从 **try** 语句中调用的函数或程序中时，**try** 语句对运行时行为没有影响。

有关其它详细信息，请参阅异常处理。



statement

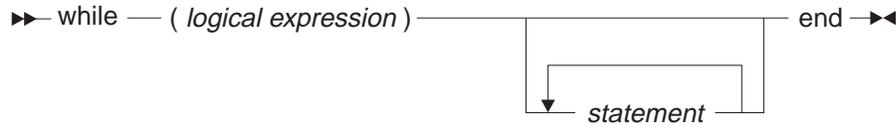
一个 EGL 语句。

OnException

在发生异常情况时运行的语句块。

while

EGL 关键字 **while** 标记在循环中运行的一组语句的开始。仅当逻辑表达式解析为 `true` 时，才发生第一次运行，且每一后续迭代都取决于相同的测试。关键字 **end** 标记 **while** 语句的结束。



logical expression

得出的值为 `true` 或 `false` 的表达式（一系列操作数和运算符）

statement

以 EGL 语言表示的语句

下面是一个示例:

```
sum = 0;
i = 1;
while (i < 4)
    sum = inputArray[i] + sum;
    i = i + 1;
end
```

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 454 页的『逻辑表达式』

第 80 页的『EGL 语句』

库（生成的输出）

Java 输出的库部件是作为 Java 类生成的。该类的名称是部件别名（如果未指定别名，则为部件名），但 EGL 会按如何为 Java 名称取别名中所述进行字符替换。

相关概念

第 131 页的『类型为 `basicLibrary` 的库部件』

第 131 页的『类型为 `basicLibrary` 的库部件』

第 678 页的『运行单元』

相关任务

第 610 页的『如何为 Java 名称取别名』

相关参考

第 592 页的『EGL 源格式的库部件』

EGL 源格式的库部件

可以在 EGL 文件中声明库部件, *EGL* 源格式对该部件作了描述。

以下是库部件的一个示例:

```
Library CustomerLib3

// Use declarations
Use StatusLib;

// Data Declarations
exceptionId ExceptionId ;

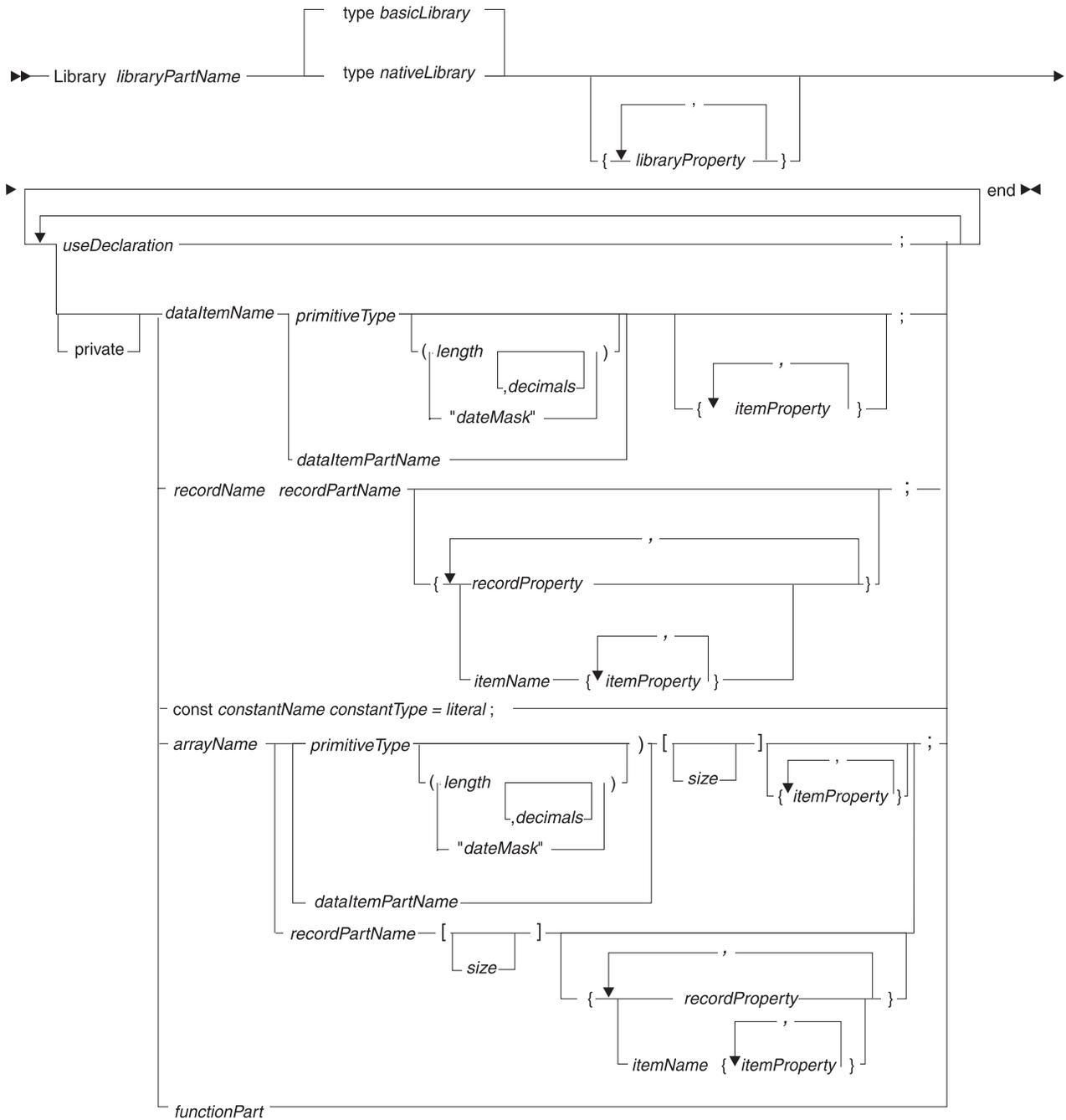
// Retrieve one customer for an email
// In: customer, with emailAddress set
// Out: customer, status
Function getCustomerByEmail ( customer CustomerForEmail, status int )
status = StatusLib.success;
try
  get customer ;
onException
  exceptionId = "getCustomerByEmail" ;
  status = sqlCode ;
end
commit();
end

// Retrieve one customer for a customer ID
// In: customer, with customer ID set
// Out: customer, status
Function getCustomerByCustomerId ( customer Customer, status int )
status = StatusLib.success;
try
  get customer ;
onException
  exceptionId = "getCustomerByCusomerId" ;
  status = sqlCode ;
end
commit();
end

// Retrieve multiple customers for an email
// In: startId
// Out: 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
```

库部件的图如下所示:



Library libraryPartName ... end

将部件标识为库部件并指定名称。如果未设置 **alias** 属性（如后文所述），则生成的库的名称是 *libraryPartName*。

有关其它规则，请参阅命名约定。

type basicLibrary, type nativeLibrary

指示库类型：

- 基本库（类型为 *basicLibrary*）包含运行时在其它 EGL 逻辑中使用的用 EGL 编写的函数和值；有关详细信息，请参阅类型为 *basicLibrary* 的库部件。

- 本机库（类型为 `nativeLibrary`）充当外部 DLL 的接口；有关详细信息，请参阅类型为 `nativeLibrary` 的库部件。

在缺省情况下，该库的类型为 `basicLibrary`。

libraryProperties

库属性如下所示：

- **alias**
- **allowUnqualifiedItemReferences**
- **callingConvention**（仅在类型为 `nativeLibrary` 的库中可用）
- **dllName**（仅在类型为 `nativeLibrary` 的库中可用）
- **handleHardIOErrors**
- **includeReferencedFunctions**
- **localSQLScope**
- **messageTablePrefix**
- **throwNrfEofExceptions**

所有属性都是可选的：

- **alias** = "*alias*" 标识合并到生成的输出的名称中的字符串。如果未设置 **alias** 属性，则会使用程序部件名。
- **allowUnqualifiedItemReferences = no** 或 **allowUnqualifiedItemReferences = yes** 指定是否允许代码引用结构项但却不包括 *container*（它是存放结构项的数据表、记录或表单）的名称。例如，请参照以下记录部件：

```
Record aRecordPart type basicRecord
  10 myItem01 CHAR(5);
  10 myItem02 CHAR(5);
end
```

以下变量基于该部件：

```
myRecord aRecordPart;
```

如果接受 **allowUnqualifiedItemReferences** 的缺省值 (*no*)，则在引用 `myItem01` 时必须指定记录名，如以下赋值所示：

```
myValue = myRecord.myItem01;
```

然而，如果将属性 **allowUnqualifiedItemReferences** 设置为 *yes*，则无需指定记录名：

```
myValue = myItem01;
```

建议您接受缺省值，这样有助于实现最佳实践。通过指定容器名，可以减少对阅读代码的人以及对 EGL 造成的歧义。

EGL 使用一组规则来确定变量名或项名所引用的内存区。有关详细信息，请参阅 *对变量和常量的引用*。

- 在类型为 `nativeLibrary` 中使用时，**callingConvention = I4GL** 指定 EGL 运行时在两种代码之间传递数据的方式：
 - 调用库函数的 EGL 代码；以及
 - 要访问的 DLL 中的函数。

现在对 **callingConvention** 可用的唯一值是 *I4GL*。有关其它详细信息，请参阅类型为 *nativeLibrary* 的库部件。

- 在类型为 *nativeLibrary* 的库中使用，**dllName** 指定 DLL 名称，这是最终名称；在部署时不能覆盖它。如果未对库属性 **dllName** 指定值，则必须在 Java 运行时属性 *vgj.defaultI4GLNativeLibrary* 中指定 DLL 名称。

有关其它详细信息，请参阅类型为 *nativeLibrary* 的库部件。

- **handleHardIOErrors = yes, handleHardIOErrors = no** 设置系统变量 **VGVar.handleHardIOErrors** 的缺省值。该变量控制在 *try* 块中的 I/O 操作发生硬错误后程序是否继续运行。该属性的缺省值为 *yes*，它将该变量设置为 1。

有关其它详细信息，请参阅 *VGVar.handleHardIOErrors* 和异常处理。

- **includeReferencedFunctions = no** 或 **includeReferencedFunctions = yes** 指示库是否包含既不在库中也不在当前库所访问的库中的每个函数的副本。缺省值为 *no*，这表示如果将作为此库一部分的所有函数都位于此库中，则可以忽略此属性。

如果该库正在使用不在库中的共享函数，则仅当将属性 **includeReferencedFunctions** 设置为 *yes* 时才有可能进行生成。

- **localSQLScope = yes, localSQLScope = no** 指示在程序和 *pageHandler* 调用期间，SQL 结果集和预编译语句的标识对于库代码是不是局部的（缺省情况下是局部的）。如果接受值 *yes*，则表示不同程序可独立使用相同的标识，并且使用该库的程序或 *pageHandler* 可独立地使用在该库中使用的相同标识。

如果指定 *no*，则标识在整个运行单元中共享。在调用库中的 SQL 语句时创建的标识在调用该库的其它代码中是可用的，尽管其它代码可使用 **localSQLScope = yes** 来阻止访问这些标识。而且，该库可以引用在调用程序或 *pageHandler* 中创建的标识，但仅当与 SQL 有关的语句已经在其它代码中运行并且其它代码没有阻止访问时才会如此。

共享 SQL 标识的影响如下所示：

- 可以在一个代码中打开某个结果集并从另一个代码中获取该结果集中的行
- 可以在一个代码中预编译某个 SQL 语句并且在另一个代码中运行该语句

在任一情况下，在程序或 *pageHandler* 访问该库时可用的标识在同一程序或 *pageHandler* 访问同一个库中的同一函数或另一函数时也是可用的。

- **msgTablePrefix = "prefix"** 指定用作消息表的数据表的名称中的第一个到第四个字符。（消息表可供作为库函数输出的表单使用。）名称中的其它字符与 *EGL* 源格式的 *DataTable* 部件中列示的其中一个本地语言代码相对应。
- **throwNrfEofExceptions = no, throwNrfEofExceptions = yes** 指定软错误是否导致抛出异常。缺省值为 *no*。有关背景知识信息，请参阅异常处理。

useDeclaration

提供对数据表或库的更简便的访问，并且是访问表单组中的表单所需的。有关详细信息，请参阅使用声明。

private

指示变量、常量或函数在库外部不可用。如果省略术语 **private**，则变量、常量或函数都是可用的。

不能对类型为 `nativeLibrary` 的库中的函数指定 **private**。

dataItemName

数据项的名称。有关命名规则，请参阅命名约定。

primitiveType

数据项的基本类型或者（对于数组）数组元素的基本类型。

length

参数的长度或者（对于数组）数组元素的长度。此长度是一个整数，它表示由 *dataItemName* 或（对于数组）*dynamicArrayName* 引用的内存区中的字符或数字的数量。

decimals

对于数字类型，可以指定 *decimals*，它是用来表示小数点后的位数的整数。最大小数位数是以下两个数字中较小的那一个：18 或声明为 *length* 的位数。小数点不与数据存储在一起。

"dateTimeMask"

对于 `TIMESTAMP` 和 `INTERVAL` 类型，可指定“*dateTimeMask*”，它会赋予日期时间值中的给定位置特别的意义（如“年份位”）。掩码不会与数据存储在一起。

dataItemPartName

对程序可视的 `dataItem` 部件的名称。有关可视性的详细信息，请参阅对部件的引用。

该部件作为格式模型，如 *Typedef* 所述。

recordName

记录的名称。有关命名规则，请参阅命名约定。

recordPartName

对程序可视的记录部件的名称。有关可视性的详细信息，请参阅对部件的引用。

该部件作为格式模型，如 *Typedef* 所述。

constantName literal

常量的名称和值。值是加引号的字符串或数字。有关命名规则，请参阅命名约定。

itemProperty

特定于项的“属性 - 值”对，如 *EGL* 属性与覆盖概述中所述。

recordProperty

特定于记录的“属性 - 值”对。有关可用属性的详细信息，请参阅您感兴趣的记录类型的相关主题。

基本记录不具有属性。

itemName

要覆盖其属性的记录项的名称。请参阅 *EGL* 属性与覆盖概述。

arrayName

记录或数据项的动态或静态数组的名称。如果使用此选项，则右边的其它符号（*dataItemPartName* 和 *primitiveType* 等等）是指数组的每个元素。

size

数组中的元素数目。如果指定元素数目，则数组是静态的；否则，数组是动态的。

functionPart

一个函数。在函数中，没有任何参数可以具有松散类型。有关详细信息，请参阅 *EGL 源格式的函数部件*。

相关概念

- 第 13 页的『EGL 项目、包和文件』
- 第 131 页的『类型为 basicLibrary 的库部件』
- 第 131 页的『类型为 basicLibrary 的库部件』
- 第 59 页的『EGL 属性概述』
- 第 20 页的『对部件的引用』
- 第 53 页的『引用 EGL 中的变量』
- 第 25 页的『Typedef』

相关参考

- 第 345 页的『EGL 源格式的基本记录部件』
 - 第 432 页的『EGL 源格式的 DataTable 部件』
 - 第 448 页的『EGL 源格式』
 - 第 86 页的『异常处理』
 - 第 481 页的『EGL 源格式的函数部件』
 - 第 488 页的『EGL 源格式的带索引记录部件』
 - 第 672 页的『输入表单』
 - 第 672 页的『输入记录』
 - 第 490 页的『I/O 错误值』
 - 第 493 页的『Java 运行时属性（详细信息）』
 - 第 603 页的『EGL 源格式的 MQ 记录部件』
 - 第 31 页的『基本类型』
 - 第 676 页的『EGL 源格式的相关记录部件』
 - 第 679 页的『EGL 源格式的串行记录部件』
 - 第 683 页的『EGL 源格式的 SQL 记录部件』
-
- 第 875 页的『使用声明』
 - 第 867 页的『handleHardIOErrors』

like 运算符

在逻辑表达式中，可将一个文本表达式与另一个字符串（称为 *like* 条件）进行比较，比较方式是从左至右比较逐个字符。此功能的用法类似于 SQL 查询中的 SQL 关键字 **like** 的用法。

下面是一个示例：

```
// variable myVar01 is the string expression
// whose contents will be compared to a like criterion
myVar01 = "abcdef";

// the next logical expression evaluates to "true"
if (myVar01 like "a_c%")
;
end
```

like 条件可以是类型为 CHAR 或 MBCHAR 的文字或项；或者类型为 UNICODE 的项。*like* 条件可包括下列任何字符：

% 充当通配符，与字符串表达式中的零个或多个字符相匹配

_ (下划线)

充当通配符，与字符串表达式中的单个字符相匹配

**** 指示下一个字符将与字符串表达式中的单个字符进行比较。反斜杠 (\) 被称为转义字符，原因是它使字符发生转义；转义字符不会与字符串表达式中的任何字符进行比较。

转义字符之前通常有百分比符号 (%)、下划线 (_) 或另一反斜杠。

将反斜杠用作转义字符（这是缺省行为）时，因为 EGL 使用同一转义字符以允许在任何文本表达式中包括引号，从而会产生问题。在 `like` 条件的上下文中，必须指定两个反斜杠，原因是运行时可用的文本就是缺少初始斜杠的文本。

建议尽量避免发生此问题。通过使用稍后示例中所示的转义子句将另一字符指定为转义字符。但是，不能使用双引号 (") 作为转义字符。

`likeCriterion` 中的另一字符是要与字符串表达式中的单个字符进行比较的文字。

以下示例显示转义子句的用法：

```
// variable myVar01 is the string expression
// whose contents will be compared to a like criterion
myVar01 = "ab%def";

// the next logical expression evaluates to "true"
if (myVar01 like "ab\\%def")
;
end

// the next logical expression evaluates to "true"
if (myVar01 like "ab+%def" escape "+")
;
end
```

相关参考

第 80 页的『EGL 语句』

第 454 页的『逻辑表达式』

第 462 页的『文本表达式』

链接属性文件（详细信息）

当生成调用 Java 程序或包装器时，可指定在运行时需要链接信息。通过将调用程序的链接选项值设置为下面的值来作出说明：

- `callLink` 元素属性 **type** 的值为 `remoteCall` 或 `ejbCall`；并且
- `callLink` 元素属性 **remoteBind** 的值为 `RUNTIME`。

链接属性文件可能是手写的，但如果（除了之前描述的设置之外）生成 Java 程序或包装器时将构建描述符选项 **genProperties** 设置为 `GLOBAL` 或 `PROGRAM`，EGL 将生成一个文件。

如果在运行时标识链接属性文件

如果在链接选项部件中将被调用程序的 `callLink` 元素属性 **remoteBind** 设置为 `RUNTIME`，则将在运行时寻找链接属性文件；但是，对于 Java 程序和 Java 包装器，文件名的源是不同的：

- Java 程序检查 Java 运行时属性 **cs0.linkageOptions.LO**，其中 *LO* 是用于生成的链接选项部件的名称。如果该属性不存在，则 EGL 运行时代码将寻找名为 **LO.properties** 的链接属性文件。同样，*LO* 是用于生成的链接选项部件的名称。

在这种情况下，如果 EGL 运行时代码寻找链接属性文件但又找不到该文件，则将在需要使用该文件的第一个 call 语句上发生错误。有关结果的详细信息，请参阅“异常处理”。

- Java 包装器将链接属性文件的名称存储在程序对象变量 *callOptions* 中，其类型为 CSOCallOptions。生成的文件名为 **LO.properties**，其中 *LO* 是用于生成的链接选项的名称。

在这种情况下，如果 Java 虚拟机寻找链接属性文件但又找不到该文件，则程序对象将抛出类型为 CSOException 的异常。

链接属性文件的格式

在运行时期间使用时，链接属性文件包括一系列条目，以处理来自正在部署的生成的 Java 程序或包装器的每个调用。

主条目的类型为 *cs0.serverLinkage*，并且可以包括可以在链接选项部件的 *callLink* 元素中设置的任何属性值对，但是下列情况例外：

- 属性 **remoteBind** 有必要是 RUNTIME，并且不应该出现
- 属性 **type** 不能是 localCall，原因是必须在生成时建立本地调用的链接

cs0.serverLinkage 条目

在最基本的情况下，链接属性文件中的每个条目的类型都是 *cs0.serverLinkage*。条目的格式如下所示：

```
cs0.serverLinkage.programName.property=value
```

programName

被调用程序的名称。如果被调用程序是由 EGL 生成的，则您指定的名称是程序部件的名称。

property

适用于 Java 程序的任何属性，**remoteBind** 和 **pgmName** 属性除外。有关详细信息，请参阅 *callLink* 元素。

value

对指定的属性有效的值。

以下是被调用程序 *XYZ* 的一个示例，其中 *xxx* 表示一个区分大小写的字符串：

```
cs0.serverLinkage.Xyz.type=ejbCall
cs0.serverLinkage.Xyz.remoteComType=TCPIP
cs0.serverLinkage.Xyz.remotePgmType=EGL
cs0.serverLinkage.Xyz.externalName=xxx
cs0.serverLinkage.Xyz.package=xxx
cs0.serverLinkage.Xyz.conversionTable=xxx
cs0.serverLinkage.Xyz.location=xxx
cs0.serverLinkage.Xyz.serverID=xxx
cs0.serverLinkage.Xyz.parmForm=COMMDATA
cs0.serverLinkage.Xyz.providerURL=xxx
cs0.serverLinkage.Xyz.luwControl=CLIENT
```

文字值 TCPIP 和 EGL 等都是不区分大小写的，并且都是有效数据的示例。

cso.application 条目

如果要创建一系列 `cso.serverLinkage` 条目来表示若干个被调用程序中的任何程序，则在这些条目前面添加一个或多个类型为 `cso.application` 的条目。在此例中，主要目的是使单个应用程序名等于多个程序名。在后续的 `cso.serverLinkage` 条目中，使用应用程序名而不使用 `programName`；于是，在 Java 运行时，那些 `cso.serverLinkage` 条目将处理对若干程序中的任何程序的调用。

`cso.application` 条目的格式如下所示：

```
cso.application.wildProgramName.appName
```

wildProgramName

有效的程序名、星号或者以有效程序名开头并且后跟一个星号。星号是等同于一个或多个字符的通配符，它可以标识一组名称。

如果 *wildProgramName* 是指由 EGL 生成的程序，则 *wildProgramName* 中包括的任何程序名都是程序部件的名称。

appName

符合 EGL 命名约定的一系列字符。*appName* 的值用于后续 `cso.serverLinkage` 条目。

以下示例显示了将星号作为通配符字符的用法。此示例中的 `cso.serverLinkage` 条目处理对其名称以 `XYZ` 开头的程序的任何调用：

```
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
```

以下示例显示了如何使用同一个 `cso.serverLinkage` 条目来处理对若干程序中的任何程序的调用，即使这些程序的名称不以相同的字符开头：

```
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
```

如果有多个 `cso.application` 条目对于程序有效，则 EGL 使用第一个适用条目。

相关概念

第 282 页的『链接选项部件』

第 330 页的『链接属性文件』

相关任务

第 285 页的『编辑链接选项部件的 `callLink` 元素』

第 321 页的『为 EGL 生成的代码设置 J2EE 运行时环境』

相关参考

第 370 页的『`callLink` 元素』

第 86 页的『异常处理』

第 493 页的『Java 运行时属性（详细信息）』

第 612 页的『命名约定』

matches 运算符

在逻辑表达式中，可将一个字符串表达式与另一个字符串（称为 *match* 条件）进行比较，比较方式是从左至右比较逐个字符。此功能的用法类似于 UNIX 或 Perl 中的正则表达式的用法。

下面是一个示例：

```
// variable myVar01 is the string expression
// whose contents will be compared to a match criterion
myVar01 = "abcdef";

// the next logical expression evaluates to "true"
if (myVar01 matches "a?c*")
;
end
```

match 条件可以是类型为 CHAR 或 MBCHAR 的文字或项；或者类型为 UNICODE 的项。*match* 条件可包括下列任何字符：

* 充当通配符，与字符串表达式中的零个或多个字符相匹配

? 充当通配符，与字符串表达式中的单个字符相匹配

[] 充当定界符，以使两个方括号之间的任一字符充当字符串表达式中的下一个字符的有效匹配。例如，*match* 条件的以下部分指示 a、b 或 c 是有效匹配：

```
[abc]
```

- 在括号定界符中创建一个范围，以使该范围内的任何字符充当字符串表达式中的下一个字符的有效匹配。例如，*match* 条件的以下部分指示 a、b 或 c 是有效匹配：

```
[a-c]
```

在括号定界符之外的连字符 (-) 就没有特殊含义。

^ 创建一条通配规则，这样的话，如果插入标记 (^) 是括号定界符中的第一个字符，则定界符之外的任何字符将充当字符串表达式中下一个字符的有效匹配。例如，*match* 条件的以下部分指示 a、b 或 c 之外的任何字符是有效匹配：

```
[^abc]
```

插入标记在下列情况下没有特殊含义：

- 在括号定界符外部
- 在括号定界符内部，但不是第一个字符

\ 指示下一个字符将与字符串表达式中的单个字符进行比较。反斜杠 (\) 被称为转义字符，原因是它使字符发生转义；转义字符不会与字符串表达式中的任何字符进行比较。

转义字符通常会加在 `match` 条件中有其它含义的字符之前；例如，加在星号 (*) 或问号 (?) 之前。

将反斜杠用作转义字符（这是缺省行为）时，因为 EGL 使用同一转义字符以允许在任何文本表达式中包括引号，从而会产生问题。在 `match` 条件的上下文中，必须指定两个反斜杠，原因是运行时可用的文本就是缺少初始斜杠的文本。

建议尽量避免发生此问题。通过使用稍后示例中所示的转义子句将另一字符指定为转义字符。但是，不能使用双引号 (") 作为转义字符。

`matchCriterion` 中的另一字符是要与字符串表达式中的单个字符进行比较的文字。

以下示例显示转义子句的用法：

```
// variable myVar01 is the string expression
// whose contents will be compared to a match criterion
myVar01 = "ab*def";

// the next logical expression evaluates to "true"
if (myVar01 matches "ab\\*[abcd][abcde][^a-e]")
;
end

// the next logical expression evaluates to "true"
if (myVar01 matches "ab+*def" escape "+")
;
end
```

相关参考

第 80 页的『EGL 语句』

第 454 页的『逻辑表达式』

第 462 页的『文本表达式』

EGL Java 运行时的消息定制

当 Java 运行时发生错误时，缺省情况下将显示 EGL 系统消息；但您可以对每条系统消息或一组系统消息指定定制消息。

需要消息时，EGL 先搜索在 Java 运行时属性 `vgj.messages.file` 中标识的属性文件。被引用文件的格式与 Java 属性文件的格式相同，如程序属性文件和当前主题中所示。

在许多情况下，系统消息包括 EGL 在运行时检索的消息插入内容的占位符。例如，如果代码将无效日期掩码提交至系统函数，消息将具有两个占位符；一个（占位符 0）表示日期掩码本身，另一个占位符（占位符 1）表示系统函数的名称。在属性文件格式中，对应缺省消息的条目如下所示：

```
VGJ0216E = {0} is not a valid date mask for {1}.
```

可更改消息的措辞以包括全部或部分占位符（以任意顺序），但不能添加占位符。有效示例如下所示：

```
VGJ0216E = 对函数 {1} 指定了无效日期掩码 {0}。
```

```
VGJ0216E = 对函数 {1} 指定了无效日期掩码。
```

如果属性 `vgj.messages.file` 中标识的文件不能打开，则表示发生了致命错误。

有关消息编号及其含义的详细信息，请参阅 *EGL Java 运行时错误代码*。

其它详细信息可在 Java 语言文档中找到：

- 有关如何处理消息以及哪些内容有效的详细信息，请参阅 Java 类 `java.text.MessageFormat` 的文档。
- 有关不能直接以 ISO 8859-1 字符编码（总是在属性文件中使用）表示的操作字符的详细信息，请参阅 Java 类 `java.util.Properties` 的文档。

相关概念

第 317 页的『程序属性文件』

相关参考

第 879 页的『EGL Java 运行时错误代码』

第 493 页的『Java 运行时属性（详细信息）』

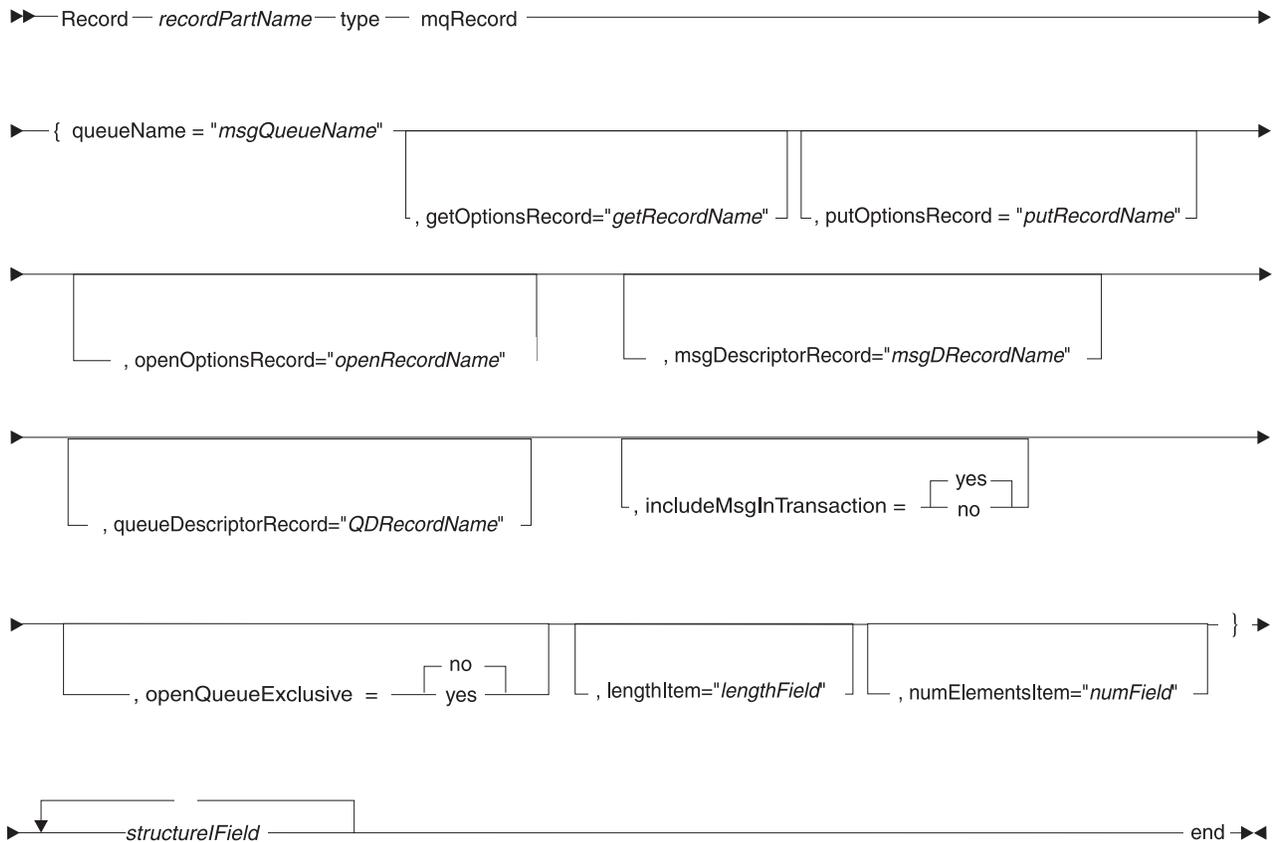
EGL 源格式的 MQ 记录部件

可以在 EGL 源文件中声明 MQ 记录部件。有关该文件的概述，请参阅 *EGL 源格式*。有关 EGL 如何与 MQSeries 进行交互的概述，请参阅 *MQSeries 支持*。

下面是 MQ 记录部件的示例：

```
Record myMQRecordPart type mqRecord
{
  queueName = "myQueue"
}
10 myField01 CHAR(2);
10 myField02 CHAR(78);
end
```

MQ 记录部件的语法图如下所示：



Record *recordPartName* mqRecord

将部件标识为具有 mqRecord 类型并指定名称。有关规则，请参阅命名约定。

queueName = "msgQueueName"

消息队列名，它是逻辑队列名并且通常不是物理队列的名称。有关输入的格式的细节信息，请参阅 MQ 记录属性。

getOptionsRecord = "getRecordName"

标识用作获取选项记录的程序变量（基本记录）。有关详细信息，请参阅 MQ 记录的选项记录。此属性以前是 **getOptions** 属性。

putOptionsRecord = "putRecordName"

标识用作放置选项记录的程序变量（基本记录）。有关详细信息，请参阅 MQ 记录的选项记录。此属性以前是 **putOptions** 属性。

openOptionsRecord = "openRecordName"

标识用作打开选项记录的程序变量（基本记录）。有关详细信息，请参阅 MQ 记录的选项记录。此属性以前是 **openOptions** 属性。

msgDescriptorRecord = "msgDRecordName"

标识用作消息描述符的程序变量（基本记录）。有关详细信息，请参阅 MQ 记录的选项记录。此属性以前是 **msgDescriptor** 属性。

queueDescriptorRecord = "QDRecordName"

标识用作队列描述符的程序变量（基本记录）。有关详细信息，请参阅 MQ 记录的选项记录。此属性以前是 **queueDescriptor** 属性。

includeMsgInTransaction = yes, includeMsgInTransaction = no

如果将此属性设置为 *yes* (缺省值), 则将每条特定于记录的消息嵌入到事务中, 并且代码可以落实或回滚该事务。有关选项的含义的详细信息, 请参阅 *MQSeries* 支持。

openQueueExclusive = no, openQueueExclusive = yes

如果将此属性设置为 *yes*, 则只有您的代码才能读取消息队列; 否则其它程序也可以读取该队列。缺省值为 *no*。此属性等同于 `MQSeries` 选项 `MQOO_INPUT_EXCLUSIVE`。

lengthItem = "lengthField"

长度字段, 如 *MQ* 记录属性中所述。

numElementsItem = "numElementsField"

元素数目字段, 如 *MQ* 记录属性中所述。

structureField

结构字段, 如 *EGL* 源格式的结构项中所述。

相关概念

第 13 页的『EGL 项目、包和文件』

第 20 页的『对部件的引用』

第 242 页的『MQSeries 支持』

第 16 页的『部件』

第 122 页的『记录部件』

第 53 页的『引用 EGL 中的变量』

第 25 页的『Typedef』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 68 页的『数组』

第 431 页的『EGL 源格式的 DataItem 部件』

第 448 页的『EGL 源格式』

第 481 页的『EGL 源格式的函数部件』

第 488 页的『EGL 源格式的带索引记录部件』

第 606 页的『MQ 记录属性』

第 612 页的『命名约定』

第 606 页的『MQ 记录的选项记录』

第 31 页的『基本类型』

第 664 页的『EGL 源格式的程序部件』

第 676 页的『EGL 源格式的相关记录部件』

第 679 页的『EGL 源格式的串行记录部件』

第 683 页的『EGL 源格式的 SQL 记录部件』

第 686 页的『EGL 源格式中的结构字段』

MQ 记录属性

本页面描述下列 MQ 记录属性:

- 队列名
- 将消息包括在事务中
- 打开输入队列以供独占使用

有关其它属性的详细信息, 请参阅下列页面:

- MQ 记录的选项记录
- 支持变长记录的属性

队列名

队列名是必需的, 并且指的是逻辑队列名, 它不能超过 8 个字符。有关输入的含义的详细信息, 请参阅与 *MQSeries* 相关的 *EGL* 关键字。

将消息包括在事务中

如果设置了将消息包括在事务中, 则将把每条特定于记录的消息嵌入在事务中, 并且代码可以落实或回滚该事务。

有关选项的含义的详细信息, 请参阅 *MQSeries* 支持。

打开输入队列以供独占使用

如果设置了打开输入队列以供独占使用, 则只有您的代码才能从消息队列中读取; 否则, 其它程序也可以从队列中读取。此属性等同于 `MQSeries` 选项 `MQOO_INPUT_EXCLUSIVE`。

相关概念

第 244 页的『与 *MQSeries* 相关的 *EGL* 关键字』

第 242 页的『*MQSeries* 支持』

第 125 页的『记录类型和属性』

相关参考

『MQ 记录的选项记录』

第 673 页的『支持变长记录的属性』

MQ 记录的选项记录

每个 MQ 记录都与 5 个选项记录相关联, *EGL* 将这些选项记录用作对 *MQSeries* 进行的隐藏调用中的自变量:

- 获取选项记录 (MQGMO)
- 放置选项记录 (MQPMO)
- 打开选项记录 (MQOO: 一条记录具有一个结构项)
- 消息描述符记录 (MQMD)
- 队列描述符记录 (MQOD)

当指定选项记录来作为 MQ 记录的属性时，引用的是将工作存储器记录部件（如 MQOD）用作 typeDef 的变量。该部件位于随产品提供的 EGL 文件中，如 MQSeries 支持中所述。您可以将记录部件复制到自己的 EGL 文件中并定制该部件，而不是按原样使用记录部件。

如果您未指示正在使用给定的选项记录，则 EGL 会构建一个缺省记录并指定值，如下列各节所述。但是，在不使用 MQ 记录的情况下访问 MQSeries 时，缺省选项记录不可用。

获取选项记录

可以根据 MQSeries 获取消息选项（MQGMO，它是 MQSeries MQGET 调用中的自变量）来创建获取选项记录。如果未声明获取选项记录，则 EGL 会自动构建一个名为 MQGMO 的缺省值，并且生成的程序将执行下列操作：

- 使用列示在数据初始化开头的值来初始化获取选项记录
- 将 OPTIONS 设置为 MQGMO_SYNCPOINT 或 MQGMO_NO_SYNCPOINT，这取决于您是否设置了 MQ 记录属性将消息包括在事务中

放置选项记录

可以根据 MQSeries 放置消息选项（MQPMO，它是 MQSeries MQPUT 调用中的自变量）来创建放置选项记录。如果未声明放置选项记录，则 EGL 会自动构建一个名为 MQPMO 的缺省值，并且生成的程序将执行下列操作：

- 使用列示在数据初始化开头的值来初始化放置选项记录
- 将 OPTIONS 设置为 MQPMO_SYNCPOINT 或 MQPMO_NO_SYNCPOINT，这取决于您是否设置了 MQ 记录属性将消息包括在事务中

打开选项记录

打开选项记录的内容确定了在调用 MQSeries 命令 MQOPEN 或 MQCLOSE 时使用的 Options 参数的值。打开选项记录部件（MQOO）是可用的，但是如果未声明基于该部件的记录，则 EGL 会自动构建一个名为 MQOO 的缺省值，如下所示：

- 对于因为 EGL add 语句而调用的 MQOPEN，生成的程序将 MQOO.OPTIONS 设置为以下值：

```
MQOO_OUTPUT + MQOO_FAIL_IF QUIESCING
```

- 当消息队列记录属性选项打开输入队列以供独占使用生效时，对于因为 EGL scan 语句而调用的 MQOPEN，生成的程序将 MQOO.OPTIONS 设置为以下值：

```
MQOO_INPUT_EXCLUSIVE + MQOO_FAIL_IF QUIESCING
```

- 当消息队列记录属性选项打开输入队列以供独占使用为无效时，在由于 EGL scan 语句而调用的 MQOPEN 上，生成的程序将 MQOO.OPTIONS 设置为以下值：

```
MQOO_INPUT_SHARED + MQOO_FAIL_IF QUIESCING
```

- 对于因为 EGL close 语句而调用的 MQCLOSE，生成的程序将 MQOO.OPTIONS 设置为以下值：

```
MQCO_NONE
```

消息描述符记录

可以根据 MQSeries 消息描述符（MQMD，它是 MQGET 和 MQPUT 调用中的参数）来创建消息描述符记录。如果未声明消息描述符记录，则 EGL 会自动构建一个名为 MQMD 的缺省值，并使用列示在数据初始化中的值来初始化该记录。

队列描述符记录

可以根据 MQSeries 对象描述符 (MQOD, 它是 MQSeries MQOPEN 和 MQCLOSE 调用中的自变量) 来创建队列描述符记录。如果未声明队列描述符记录, 则 EGL 会自动构建一个名为 MQOD 的缺省值, 并且生成的程序将执行下列操作:

- 使用列示在数据初始化的开头中的值来初始化队列描述符记录
- 将该记录中的 OBJECTTYPE 设置为 MQOT_Q
- 将 OBJECTMGRNAME 设置为系统字 **record.resourceAssociation** 中指定的队列管理器名; 但是如果 **record.resourceAssociation** 未引用该队列管理器名, 则 OBJECTMGRNAME 没有值
- 将 OBJECTNAME 设置为 **record.resourceAssociation** 中的队列名

相关概念

第 246 页的『直接 MQSeries 调用』

第 244 页的『与 MQSeries 相关的 EGL 关键字』

第 242 页的『MQSeries 支持』

相关参考

第 430 页的『数据初始化』

第 786 页的『recordName.resourceAssociation』

第 606 页的『MQ 记录属性』

为名称取别名

如果用在 Java 输出中无效的名称, 则生成器将在生成的代码中创建和使用该名称的别名, 原因为下列其中一项:

- 允许标识字符中的差别
- 长度局限性差别
- 对大小写字符的支持的差别
- 使用的字是生成的语言中的保留字
- 使用的字与名称别名语法有冲突 (例如, 由于 **class\$** 是 Java 生成中的 **class** 的别名, 所以将为 **class\$** 取别名)

可以通过用一组有效字符替代无效字符、通过将过长的名称截断、通过将前缀或后缀添加至名称或通过生成完全不同的名称 (如 **EZE00123**) 来生成别名。

相关概念

相关任务

第 127 页的『创建 EGL 程序部件』

相关参考

第 610 页的『如何为 Java 名称取别名』

第 610 页的『如何为 Java 包装器名取别名』

第 612 页的『命名约定』

对 JSP 文件和生成的 Java bean 中的 EGL 标识的更改

按照命名约定中详细描述的规则，对 PageHandler 函数、记录和项指定名称。但是，EGL 在 JSP 文件和派生自 PageHandler 的 Java bean 中创建 Java 标识时使用这些名称的变体。如果使用源选项卡编辑 JSP 文件、使用“属性”视图或者完全脱离启用 EGL 的工具进行工作，则需要知道这些变体。

变体如下所示：

- 字母 *EGL* 在 PageHandler 记录、项和函数的名称之前。这一变体可使您避免 Java 运行时环境中出现错误，这是 Java bean 规范与 EGL 中的命名约定之间的差别造成的。
- 在某些情况下，会向与将特定种类的输出控制绑定的变量的名称添加后缀：
 - 如果将某个项与布尔值复选框绑定，Java 标识将包括 *AsBoolean* 后缀
 - 如果将某个项与选择控件（列表框、组合框、单选按钮组或复选框组）绑定并在 JavaServer Faces selectItems 标记中引用该项，则 Java 标识包括 *AsSelectItemsList* 后缀
 - 如果将某个项与 JavaServer Faces 数据表中的复选框绑定（具体地说，如果在 inputRowSelect 标记中引用了该项），Java 标识将包括 *AsIntegerArray* 后缀

除了之前列示的变体之外，EGL 还尝试创建与 PageHandler 中的名称完全匹配的标识。

考虑 PageHandler *myJSP*，它包括 *myItem* 变量。如果将该变量与布尔值复选框绑定，JSP 文件将引用 Java bean 属性 *myJSP.EGLmyItemAsBoolean* 并且 Java bean getter 和 setter 函数的命名如下所示：

- *getEGLmyItemAsBoolean*
- *setEGLmyItemAsBoolean*

JSP 文件中的布尔值复选框标记的源代码如下所示：

```
<h:selectBooleanCheckbox styleClass="selectBooleanCheckbox"
    id="checkbox1" value="{myJSP.EGLmyItemAsBoolean}">
</h:selectBooleanCheckbox>
```

EGL 避免生成在 Java 中无效的名称；有关详细信息，请参阅[如何为 Java 名称取别名](#)。

相关概念

第 177 页的『PageHandler』

相关任务

第 181 页的『创建 EGL 字段并将其与 Faces JSP 相关联』

第 182 页的『将 EGL 记录与 Faces JSP 相关联』

第 184 页的『对 PageHandler 代码使用“快速编辑”视图』

相关参考

第 610 页的『如何为 Java 名称取别名』

第 612 页的『命名约定』

第 175 页的『EGL 的 Page Designer 支持』

如何为 Java 名称取别名

当对部件指定名称时，该名称必须是有效的 Java 标识，但可以在部件名中使用连字符或减号 (-)。但是，连字符不能是部件名的第一个字符。

如果选择作为 Java 关键字的名称或者包含美元符号 (\$) 或连字符或减号的名称，则部件名将不会与生成的输出中的名称相匹配。别名判别机制将对每个作为 Java 关键字的部件名自动追加美元符号。如果指定包含一个或多个美元符号或连字符的名称，则别名判别机制会将每个符号替换为 Unicode 值，如下所示：

```
$ $0024  
- $002d
```

例如，为名为 **class** 的项取别名 **class\$**，为名为 **class\$** 的项取别名 **class\$0024**。

用于声明部件名的大小写必须保留。程序 XYZ 和 xyz 分别是在 XYZ.java 和 xyz.java 中生成的。在 Windows 2000/NT/XP 上，如果生成到仅名称大小写不同的相同目录部件中，就会覆盖旧文件。

EGL 包名始终被转换为小写的 Java 包名。

最后，如果程序名、PageHandler 名或库名与 Java 系统包 java.lang 中的类名相匹配，则对类名追加美元符号：Object 变为 Object\$，Error 变为 Error\$，依此类推。

有关 EGL 如何在 JSP 文件和派生自 PageHandler 的 Java bean 中创建 Java 标识的详细信息，请参阅对 JSP 文件中的 EGL 标识和生成的 Java bean 的更改。

相关概念

第 608 页的『为名称取别名』

相关参考

第 609 页的『对 JSP 文件和生成的 Java bean 中的 EGL 标识的更改』

如何为 Java 包装器名取别名

EGL 生成器应用下列规则来为 Java 包装器名称取别名：

1. 如果 EGL 名称是全大写的，则将它转换为小写。
2. 如果名称是类名或方法名，则使第一个字符为大写。（例如，**x** 的 **getter** 方法是 **getX()** 而不是 **getx()**。）
3. 删除每个下划线 (_) 和连字符 (-)。（如果使用 VisualAge Generator 兼容性方式，则连字符在 EGL 名称中有效。）如果一个字母跟着下划线或连字符，则将该字符更改为大写字符。
4. 如果名称是使用句点 (.) 作为分隔符的限定名，则将每个句点替换为下划线，并在该名称开头添加下划线。
5. 如果名称中包含美元符号 (\$)，则将美元符号替换为两个下划线，并在该名称开头添加下划线。
6. 如果名称是 Java 关键字，则在该名称开头添加下划线。
7. 如果名称是 *（一个星号，它表示填充符项），则将第一个星号重命名为 **Filler1**，将第二个星号重命名为 **Filler2**，依此类推。

另外，特殊规则适用于程序包装器、记录包装器和具有子结构的数组项的 Java 包装器类名。余下各节讨论这些规则并给出一个示例。一般情况下，如果生成的包装器类中的字段中存在命名冲突，则使用限定名来确定类和变量名。如果仍然没有解决冲突，则会在生成时抛出异常。

程序包装器类

记录参数包装器是使用以上适用于类型定义名的规则来命名的。如果记录包装器类名与程序类名或程序包装器类名冲突，则会将 **Record** 添加至记录包装器类名的末尾。

用于变量名的规则如下所示：

1. 记录参数变量是使用以上适用于参数名的规则命名的。因此，**get()** 和 **set()** 方法包含这些名称而不是类名。
2. 按照以上提供的规则，**get** 和 **set** 方法被命名为 **get** 或 **set** 后跟参数名。

记录包装器类

用于具有子结构的数组项类名的规则如下：

1. 具有子结构的数组项成为记录包装器类的内部类，类名是通过将以上规则应用于项名来派生的。如果此类名与包含的记录类名冲突，则会将 **Structure** 追加至项类名。
2. 如果任何项类名互相冲突，则会使用限定项名。

用于 **get** 和 **set** 方法名的规则如下所示：

1. 按照以上提供的规则，这些方法被命名为 **get** 或 **set** 后跟项名。
2. 如果任何项名互相冲突，则会使用限定项名。

具有子结构的数组项类

用于具有子结构的数组项类名的规则如下：

1. 具有子结构的数组项成为为包含具有子结构的数组项而生成的包装器类的内部类，类名是通过将以上规则应用于项名来派生的。
2. 如果此类名与包含的具有子结构的数组项类名冲突，则会将 **Structure** 追加至项类名。

用于 **get** 和 **set** 方法名的规则如下所示：

1. 按照以上提供的规则，这些方法被命名为 **get** 或 **set** 后跟项名。
2. 如果任何项名互相冲突，则会使用限定项名。

示例

以下样本程序和生成的输出显示在包装器生成期间的期望结果：

样本程序：

```
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

```

生成的输出:

生成的输出的名称

输出	名称
程序包装器类	WrapperAliasWrapper , 包含字段 param1 , 它是记录包装器类 RecordA 的实例
参数包装器类	<p>RecordA, 可通过下列方法来访问:</p> <ul style="list-style-type: none"> • getItemA (来自 itemA) • getItemB (来自第一个 item-b) • get_Item_C (来自 item\$C) • get_Static (来自 static) • get_ItemC_itemB (来自 itemC 中的 itemB) • getItemD (来自 itemD) • getArrayItem (来自 arrayItem) <p>ArrayItem 是 RecordA 的内部类, 它包含可通过 getInnerItem1 和 getInnerItem2 访问的字段。</p>

相关概念

第 400 页的『与 VisualAge Generator 的兼容性』

第 274 页的『Java 包装器』

第 608 页的『为名称取别名』

相关任务

第 273 页的『生成 Java 包装器』

相关参考

第 502 页的『Java 包装器类』

『命名约定』

第 615 页的『Java 包装器生成输出』

命名约定

本页描述用于命名部件和变量以及用于对属性（例如，**文件名**）赋值的规则。有关逻辑部件可以如何引用内存区的详细信息，请参阅对变量和常量的引用以及数组。

在 EGL 中有三个标识类别:

- EGL 部件和变量名，如后文所述。
- 在部件声明或变量声明中作为属性值指定的外部资源名称。这些名称表示特殊的大小写，并且命名约定取决于运行时系统的约定。
- EGL 包名，如 com.mycom.mypack。在这种情况下，每个字符序列都通过句点与下一个字符序列隔开，每个序列都遵循 EGL 部件名的命名约定。有关包名与文件结构的关系的详细信息，请参阅 *EGL 项目、包和文件*。

EGL 部件名或变量名是 1 到 128 个字符的一串字符。除了上面提到的几点之外，名称还必须以 Unicode 字母或下划线开头，并且可以包含其它 Unicode 字母以及数字和货币符号。存在其它有效限制：

- 开头字符不能为混合大小写的 EZE。
- 名称不能包含嵌入空格，也不能是 EGL 保留字

下列特殊注意事项适用于部件：

- 在记录部件中，逻辑文件或队列的名称不能超过 8 个字符
- 在各个部件中，*alias* 包含在生成的输出文件和 Java 类的名称中。如果未指定外部名，则使用程序部件的名称，但会将该名称截断为运行时环境所允许的最大字符数（必要时）。

如果代码与 VisualAge Generator 兼容，则下列规则也适用于部件名和变量名，但不影响包名：

- 名称的初始字符可以是 @ 符
- 后续字符可以包括 @ 符、连字符 (-) 和磅符 (#)

相关概念

- 第 400 页的『与 VisualAge Generator 的兼容性』
- 第 13 页的『EGL 项目、包和文件』
- 第 608 页的『为名称取别名』
- 第 53 页的『引用 EGL 中的变量』

相关参考

- 第 68 页的『数组』
- 第 609 页的『对 JSP 文件和生成的 Java bean 中的 EGL 标识的更改』
- 第 444 页的『EGL 保留字』
- 第 451 页的『EGL 系统限制』

运算符和优先顺序

下表按优先顺序的降序列示了 EGL 运算符。除了一元的正 (+)、负 (-) 和非 (!) 之外，每个运算符都使用两个操作数。

运算符（用逗号分隔）	运算符类型	含义
+ 和 -	数字，一元	一元的正 (+) 或负 (-) 是操作数或用圆括号括起来的表达式前面的符号，而不是两个表达式之间的运算符。
**	数字	** 是 <i>toThePowerOfInteger</i> 运算符，它表示某个数字的指定幂。例如， <code>c = a**b</code> 导致 <code>c</code> 被赋值为 (a^b) 。第一个操作数（上述示例中的 <code>a</code> ）不能为负值。第二个操作数（上述示例中的 <code>b</code> ）必须为整数，或者是精度为 0 的数字字段。第二个操作数可以为正数、负数或 0。

运算符 (用逗号分隔)	运算符类型	含义
, /, %	数字	乘 () 和整数除法 (/) 的优先顺序相同。整数的除法将保留小数值 (如果有的话); 例如, 7/5 结果为 1.4。 % 是余数运算符, 它解析为当两个操作数或数字表达式中的第一个除以第二个时所获得的模数; 例如 7%5 将得到 2。
+ 和 -	数字	加 (+) 和减 (-) 的优先顺序相同。
=	数字或字符串	= 号是赋值运算符, 它将数字或字符值从一个表达式或操作数复制到另一个操作数中。
!	逻辑, 一元	! 是非运算符, 它解析为布尔值 (true 或 false), 该布尔值与紧随其后的逻辑表达式的值相反。必须用圆括号将后续表达式括起来。
==, !=, <, >, <=, >=, in, is, not	用于比较的逻辑运算符	用于比较的逻辑运算符的优先顺序相同, 并在有关逻辑表达式的页面中进行了描述。每个运算符都解析为 true 或 false。
&&	逻辑	&& 是与运算符, 它表示“两者都必须为 true”。如果运算符前面的逻辑表达式为 true 且运算符后面的逻辑表达式解析为 true, 则该运算符解析为 true; 否则, && 将解析为 false。
	逻辑	是或运算符, 它表示“其中一个或两者”。如果运算符前面的逻辑表达式为 true 或者运算符后面的逻辑表达式为 true 或者这两者都为 true, 则该运算符将解析为 true; 否则 将解析为 false。

通过使用圆括号将一个表达式与另一个表达式隔开, 可以覆盖一般优先顺序 (也称为运算顺序)。表达式中具有相同优先顺序的运算是按从左到右的顺序来求值的。

相关参考

- 第 486 页的『in 运算符』
- 第 454 页的『逻辑表达式』
- 第 461 页的『数字表达式』
- 第 31 页的『基本类型』
- 第 462 页的『文本表达式』

Java 程序生成输出

Java 服务器程序生成输出如下所示:

- 构建规划 (如果省略构建描述符选项 **genProject** 的话)
- Java 源代码 (请参阅 *Java 程序*、*PageHandler* 和库)
- 准备和运行程序所需的相关对象 (请参阅 *Java 程序*、*PageHandler* 和库)
- J2EE 环境文件
- 程序属性文件
- 结果文件 (如果省略了 **genProject** 的话)

可以使用 EGL 生成器来生成整个 Java 程序。程序和记录是作为独立 Java 类生成的。函数是作为程序中的方法生成的。数据项和结构项是作为它们所属的记录或程序类的字段生成的。

下表显示了生成的各种类型的 Java 部件的名称:

生成的 Java 部件的名称

部件类型和名称	生成的内容
名为 P 的程序	P.java 中名为 P 的类
程序 P 中名为 F 的函数	P.java 中名为 \$funcF 的 P 类的方法
名为 R 的记录	EzeR.java 中名为 EzeR 的类
名为 R 的基本记录, 函数 F 的参数	Eze\$paramR.java 中名为 Eze\$paramR 的类
名为 L 的链接选项部件	名为 L.properties 的链接属性文件
名为 Lib 的库	Lib.java 中名为 Lib 的类
名为 DT 的数据表	EzeDT.java 中名为 EzeDT 的类
名为 F 的表单	EzeF.java 中名为 EzeF 的类
名为 FG 的表单组	FG.java 中名为 FG 的类

1. 对于指示的部件类型, 可能存在两个或更多个同名部件。在该情况下, 第二个部件的名称具有附加的后缀 \$v2。第三个部件的名称将具有 \$v3 后缀, 第四个将具有 \$v4 等等。

如果命名格式会导致两个名称相同, 则 EGL 会对在第一个文件之后生成的每个文件添加后缀。后缀如下所示:

\$vn

其中

n 是从 2 开始按顺序指定的整数。

相关概念

第 297 页的『构建规划』

第 324 页的『J2EE 环境文件』

第 297 页的『Java 程序、PageHandler 和库』

第 330 页的『链接属性文件』

第 317 页的『程序属性文件』

第 298 页的『结果文件』

相关参考

第 370 页的『callLink 元素』

Java 包装器生成输出

Java 包装器生成输出如下所示:

- 构建规划 (如果省略构建描述符选项 **genProject** 的话)
- 用于包装对 Java 服务器程序进行的调用的 **JavaBeans™** (请参阅 *Java 包装器*)
- EJB 会话 bean (在特定情况下); 有关详细信息, 请参阅链接选项部件中有关 **callLink** 元素的说明
- 结果文件 (如果省略了 **genProject** 的话)

可以使用生成的 bean 来包装从非 EGL Java 类（如 servlet、EJB 或 Java 应用程序）对服务器程序的调用。将生成下列类型的类：

- 用于服务器的 bean
- 用于记录参数的 bean
- 用于记录数组行的 bean

下表显示了生成的各种类型的 Java 包装器部件的名称：

生成的 Java 包装器部件的名称

部件类型和名称	生成的内容
名为 P 的程序	PWrapper.java 中名为 PWrapper 的类
用作参数的名为 R 的记录	R.java 中名为 R 的类
用作参数的记录 R 中具有子结构的区域 S	R.java 中名为 R.S 的类
名为 L 的链接选项部件	名为 L.properties 的链接属性文件

1. 对于指示的部件类型，可能存在两个或更多个同名部件。在该情况下，第二个部件的名称具有附加的后缀 \$v2。第三个部件的名称将具有 \$v3 后缀，第四个将具有 \$v4 等等。

当请求将某个程序部件生成成为 Java 包装器时，EGL 会为下列每个可执行文件生成 Java 类：

- 程序部件
- 被声明为程序参数的每个记录
- 会话 bean（如果指定链接选项部件并且生成的程序的 **callLink** 元素具有链接类型 **ejbCall** 的话）

另外，为每个记录生成的类包括具有下列特征的每个结构项的内部类（或内部类中的类）：

- 位于那些记录的其中一个的内部结构中
- 具有至少一个下级结构项；即，具有子结构
- 是一个数组；在这种情况下，是一个具有子结构的数组

生成的每个类都存储在文件中。EGL 生成器创建在 Java 包装器中使用的名称，如下所示：

- 将名称转换为小写。
- 删除每个连字符或减号 (-) 或下划线 (_)。跟在连字符或下划线后面的字符更改为大写。
- 当将该名称用作类名或在方法名中使用时，将第一个字符转换回大写。

如果程序的其中一个参数是记录，则 EGL 还会为该变量生成包装器类。如果程序 Prog 具有 typeDef 命名为 Rec 的记录参数，则该参数的包装器类将名为 Rec。如果参数的 typeDef 与程序同名，则该参数的包装器类将具有“Record”后缀。

如果记录参数具有数组项且该项下面具有其它项，则生成器还会生成包装器。这种具有子结构的数组包装器成为记录包装器的内部类。在大多数情况下，Rec 中名为 Alttem 的具有子结构的数组项将由名为 Rec.Alttem 的类包装。记录可包含两个同名的具有子结

构的数组项，在这种情况下，项包装器是使用该项的限定名来命名的。如果第一个 `Item` 的限定名是 `Top1.Item` 且第二个的限定名是 `Top2.Middle2.Item`，则类将被命名为 `Rec.Top1$Item` 和 `Rec.Top2$middle2$Item`。如果具有子结构的数组的名称与程序的名称相同，则具有子结构的数组的包装器类将具有 `Structure` 后缀。

设置和获取低层项的值的方法被生成到每个记录包装器和具有子结构的数组包装器中。如果记录或具有子结构的数组中有两个同名的低层项，则生成器使用在前面段落中描述的限定名方案。

其它方法被生成到 `SQL` 记录变量的包装器中。对于记录变量中的每一项，生成器会创建获取和设置其空指示符值的方法以及获取和设置其 `SQL` 长度指示符的方法。

一旦编译了类，就可以使用 `Javadoc` 工具来构建 `classname.html` 文件。HTML 文件描述类的公共接口。如果使用 `Javadoc` 创建的 HTML 文件，则确保它是 `EGL Java` 包装器。从 `VisualAge Generator Java` 包装器生成的 HTML 文件不同于从 `EGL Java` 包装器生成的那些 HTML 文件。

示例

以下是一个带有具有子结构的数组的记录部件的示例：

```
Record myRecord type basicRecord
  10 MyTopStructure[3];
  15 MyStructureItem01 CHAR(3);
  15 MyStructureItem02 CHAR(3);
end
```

对于程序部件，输出文件命名如下所示：

```
aliasWrapper.java
```

其中

alias

是别名（如果有的话），它是在程序部件中指定的。如果未指定外部名，则使用程序部件的名称。

对于声明为程序参数的每个记录，输出文件命名如下所示：

```
recordName.java
```

其中

recordName

是记录部件的名称

对于具有子结构的数组，内部类的名称和位置取决于数组名在记录中是否唯一：

- 如果数组名在记录中是唯一的，则内部类位于记录类内且命名如下所示：

```
recordName.siName
```

其中

recordName

是记录部件的名称

siName

是数组的名称

- 如果数组名在记录中不是唯一的，则内部类的名称取决于数组的标准名称，用美元符号 (\$) 和下划线 (_) 的组合来分隔限定符。例如，如果数组在记录的第三层中，则生成的类是记录类的内部类且命名如下所示：

Topname\$_Secondname\$_Siname

其中

Topname

是顶层结构项的名称

Secondname

是第二层结构项的名称

Siname

是具有子结构的数组项的名称

如果另一个同名数组直接作为最高层记录的下级，则内部类也处于记录类中且命名如下所示：

Topname\$_Siname

其中

Topname

是最高层结构项的名称

Siname

是具有子结构的数组项的名称

最后，请参照以下情况：具有子结构的数组的名称在记录中不是唯一的，并且该数组是另一个具有子结构的数组的下级，后一个数组的名称在记录中也不是唯一的。下级数组的类是作为内部类的内部类生成的。

当生成 Java 包装器时，还会生成 Java 属性文件和链接属性文件（如果请求在运行时设置该链接选项的话）。

相关概念

第 297 页的『构建规划』

第 286 页的『EJB 会话 bean』

第 274 页的『Java 包装器』

第 282 页的『链接选项部件』

第 330 页的『链接属性文件』

第 298 页的『结果文件』

相关任务

第 273 页的『生成 Java 包装器』

相关参考

第 370 页的『callLink 元素』

第 502 页的『Java 包装器类』

EGL 源格式的 PageHandler 部件

可以在 EGL 文件中声明 pageHandler 部件, *EGL* 项目、包和文件对该部件作了描述。此部件是可生成部件, 这意味着它必须位于文件的顶层, 并且必须与文件同名。

以下是 pageHandler 部件的一个示例:

```
// Page designer requires that all pageHandlers
// be in a package named "pagehandlers".
package pagehandlers ;

PageHandler ListCustomers
  {onPageLoadFunction="onPageLoad"}

  // Library for customer table access
  use CustomerLib3;

  // List of customers
  customerList Customer[] {maxSize=100};

Function onPageLoad()

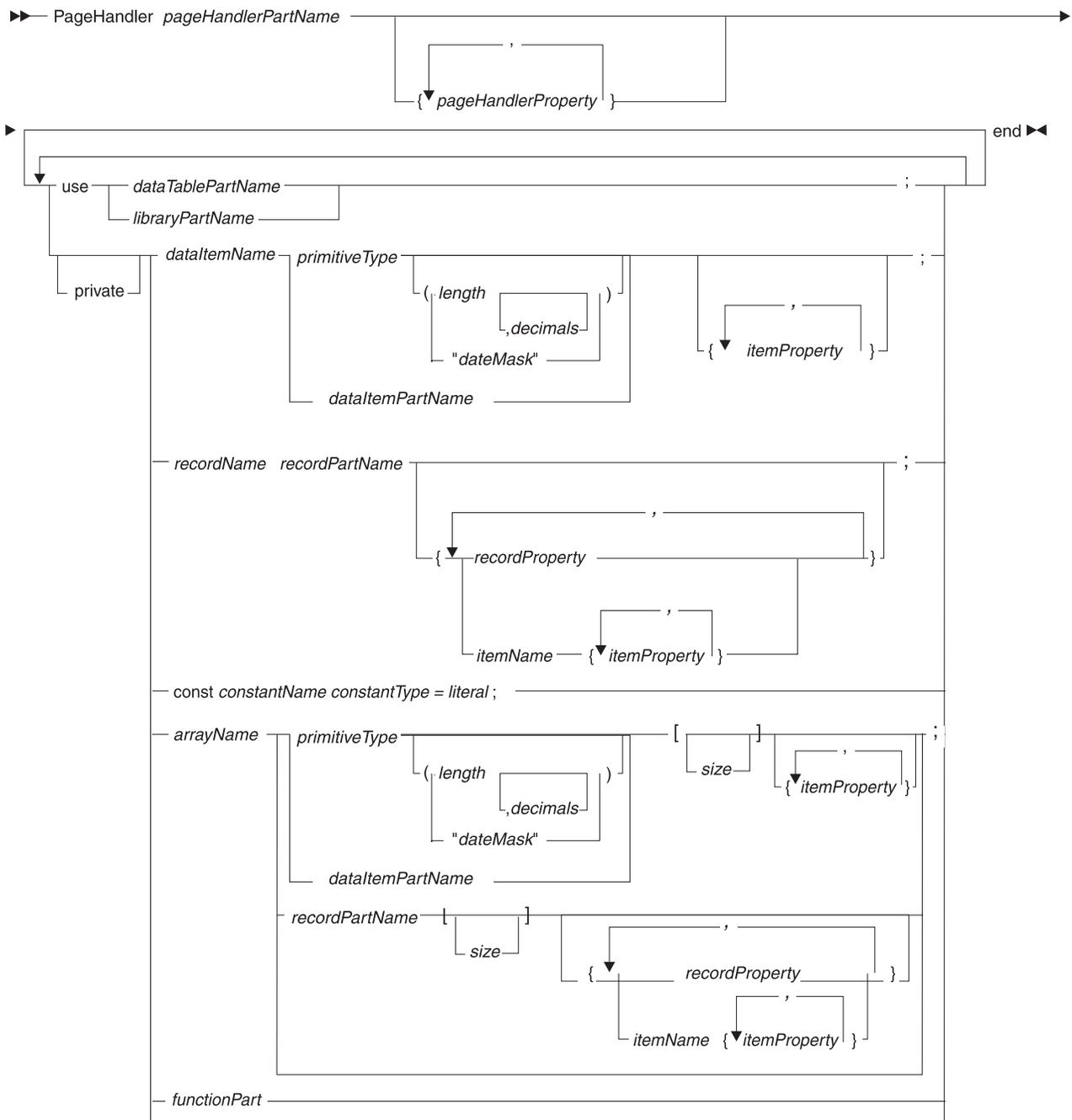
  // Starting key to retrieve customers
  startkey CustomerId;

  // Result from library call
  status int;

  // Retrieve up to 100 customer records
  startKey = 0;
  CustomerLib3.getCustomersByCustomerId(startKey, customerList, status);
  if ( status != 0 && status != 100 )
    setError("Retrieval of Customers Failed.");
  end
end

Function returnToIntroductionClicked()
  forward to "Introduction";
end
End
```

pageHandler 部件的图如下所示:



PageHandler *pageHandlerPartName* ... **end**

将部件标识为 PageHandler 并指定部件名。有关命名规则，请参阅命名约定。

pageHandlerProperty

PageHandler 部件属性，如 PageHandler 部件属性中所示。

use *dataTablePartName*, **use** *libraryPartName*

使用声明，它简化对数据表或库的访问。有关详细信息，请参阅使用声明。

private

指示变量、常量或函数对于显示 Web 页面的 JSP 来说不可用。如果省略术语 private，则可以将变量、常量或函数与 Web 页面上的控件绑定。

dataItemName

数据项（变量）的名称。有关规则，请参阅命名约定。

primitiveType

对数据项指定的基本类型。

length

结构项的长度，它是一个整数。基于结构项的内存区的值包括指定的字符或数字的数量。

decimals

对于数字类型（BIN、DECIMAL、NUM、NUMC 或 PACF），可以指定 *decimals*，它是用来表示小数点后的位数的整数。最大小数位数是以下两个数字中较小的那一个：18 或声明为 *length* 的位数。小数点不与数据存储在一起。

dataItemPartName

dataItem 部件的名称，该部件是数据项的格式模型，如 *typeDef* 所述。*dataItem* 部件必须对 *pageHandler* 部件可视，如对部件的引用中所述。

itemProperty

项属性。有关详细信息，请参阅页项属性。

recordName

记录（变量）的名称。有关规则，请参阅命名约定。

recordPartName

记录部件的名称，该部件是记录的格式模型，如 *typeDef* 所述。记录部件必须对 *pageHandler* 部件可视，如对部件的引用中所述。

recordProperty

记录属性覆盖。有关记录属性的详细信息，根据 *recordPartName* 中的记录类型的不同，请参阅下列其中一项描述：

- EGL 源格式的基本记录部件
- EGL 源格式的带索引记录部件
- EGL 源格式的 MQ 记录部件
- EGL 源格式的相关记录部件
- EGL 源格式的串行记录部件
- EGL 源格式的 SQL 记录部件

itemName

您打算覆盖其属性的记录项的名称。

itemProperty

项属性覆盖。有关详细信息，请参阅 *EGL 属性与覆盖概述*。

constantName literal

常量的名称和值。有关规则，请参阅命名约定。

arrayName

记录或数据项的动态或静态数组的名称。如果使用此选项，则右边的其它符号（*dataItemPartName* 和 *primitiveType* 等等）是指数组的每个元素。

functionPart

嵌入的函数。有关语法的详细信息，请参阅 *EGL 源格式的函数部件*。

相关概念

- 第 13 页的『EGL 项目、包和文件』
- 第 59 页的『EGL 属性概述』
- 第 177 页的『PageHandler』
- 第 20 页的『对部件的引用』
- 第 53 页的『引用 EGL 中的变量』
- 第 25 页的『Typedef』

相关参考

- 第 86 页的『异常处理』
- 第 481 页的『EGL 源格式的函数部件』
- 第 612 页的『命名约定』
- 第 624 页的『PageHandler 字段属性』
- 『PageHandler 部件属性』
- 第 31 页的『基本类型』
- 第 675 页的『EGL 中的引用兼容性』
- 第 831 页的『setError()』
- 第 875 页的『使用声明』

PageHandler 部件属性

PageHandler 的属性是可选的，如下所示（特定于 PageHandler 字段的属性除外）：

alias = "alias"

一个字符串，它包含在生成的输出的名称中。如果未指定别名，则改为使用 PageHandler 部件名。

allowUnqualifiedItemReferences = no, allowUnqualifiedItemReferences = yes

指定是否允许代码引用结构项但不包括容器的名称，该容器是存放结构项的数据表、记录或表单。例如，请参照以下记录部件：

```
Record aRecordPart type basicRecord
  10 myItem01 CHAR(5);
  10 myItem02 CHAR(5);
end
```

以下变量基于该部件：

```
myRecord aRecordPart;
```

如果接受 **allowUnqualifiedItemReferences** 的缺省值 (*no*)，则在引用 myItem01 时必须指定记录名，如以下赋值所示：

```
myValue = myRecord.myItem01;
```

然而，如果将属性 **allowUnqualifiedItemReferences** 设置为 *yes*，则无需指定记录名：

```
myValue = myItem01;
```

建议您接受缺省值，这样有助于实现最佳实践。通过指定容器名，可以减少对阅读代码的人以及对 EGL 造成的歧义。

EGL 使用一组规则来确定变量名或项名所引用的内存区。有关详细信息，请参阅对 *变量和常量的引用*。

handleHardIOErrors = yes, handleHardIOErrors = no

设置系统变量 `VGVar.handleHardIOErrors` 的缺省值。该变量控制在 `try` 块中的 I/O 操作发生硬错误后程序是否继续运行。该属性的缺省值为 `yes`，它将该变量设置为 1。

有关其它详细信息，请参阅 `VGVar.handleHardIOErrors` 和异常处理。

includeReferencedFunctions = no, includeReferencedFunctions = yes

指示 `PageHandler` bean 是否包含既不在 `PageHandler` 中也不在 `PageHandler` 访问的库中的每个函数的副本。缺省值为 `no`，这表示如果在开发时按照建议遵守以下惯例，则可以忽略此属性：

- 将共享函数放在库中
- 将非共享函数放在 `PageHandler` 中

如果正在使用不在库中的共享函数，则仅当将属性 `includeReferencedFunctions` 设置为 `yes` 时才有可能进行生成。

localSQLScope = no, localSQLScope = yes

指示 SQL 结果集和预编译语句的标识对于 `pageHandler` 是不是局部的（在缺省情况下是局部的）。如果接受值 `yes`，则表示从 `pageHandler` 调用的不同程序可独立使用相同的标识。

如果指定 `no`，则标识在整个运行单元中共享。在当前代码中创建的标识在其它地方也是可用的，尽管其它代码可使用 `localSQLScope = yes` 来阻止访问这些标识。而且，当前代码可以引用在其它位置创建的标识，但仅当其它代码已经运行并且没有阻止访问时才会如此。

共享 SQL 标识的影响如下所示：

- 可以在一个 `pageHandler` 或被调用程序中打开某个结果集并从另一个代码中获取该结果集中的行
- 可以在一个代码中预编译某个 SQL 语句并且在另一个代码中运行该语句

msgResource = "logicalName"

标识显示错误消息时要使用的 Java 资源束或属性文件。资源束或属性文件的内容由一组键和相关的值组成。

当程序对 EGL 系统函数 `sysLib.setError` 的调用使用了特定值的键时，作为对该调用的响应，将显示该值。

onPageLoadFunction = "functionName"

`PageHandler` 函数的名称，当相关 JSP 最初显示 Web 页面时，该 `PageHandler` 函数接收控制权。此函数可用于设置页中显示的数据的初始值。此属性以前是 `onPageLoad` 属性。

传递至函数的自变量在引用时必须兼容的，如 `EGL` 中的引用兼容性中所述。

scope = session, scope = request

指定在将 `pageHandler` 数据发送至 Web 页面后会发生的操作：

- 如果作用域设置为 `session`（这是缺省值），`pageHandler` 变量值将保留在整个用户会话中，并且用户以后在访问同一 `pageHandler` 时不会重复调用 `OnPageLoad` 函数
- 如果作用域设置为 `request`，`pageHandler` 变量值将丢失，并且用户在访问同一 `PageHandler` 时将重新调用 `OnPageLoad` 函数

建议显式设置此属性以表示您的决定，这会极大地影响 Web 应用程序的设计和操作系统。

throwNrfEofExceptions = no, throwNrfEofExceptions = yes

指定软错误是否导致抛出异常。缺省值为 *no*。有关背景知识信息，请参阅异常处理。

title = "literal"

title 属性是一个绑定属性，它表示当您在 Page Designer 中工作时，该属性的赋值将被用作缺省值。该属性指定页面的标题。

literal 是加引号的字符串。

validationBypassFunctions = ["functionNames"]

标识一个或多个事件处理程序，它们是与 JSP 中的按钮控件相关联的 PageHandler 函数。用逗号将每个函数名与下一个函数名隔开。

如果在此上下文中指定事件处理程序，则当用户单击属于事件处理程序或与事件处理程序相关的按钮或超文本链接时，EGL 运行时将跳过输入字段和页验证。此属性对于保留用于结束当前 PageHandler 处理并将控制权立即转移至另一个 Web 资源的用户操作而言很有用。

validatorFunction = "functionName"

标识 PageHandler 验证器函数，在调用了所有项验证器之后，将调用这个 PageHandler 验证器函数，如使用 EGL 构建的 Web 应用程序中的验证中所述。此属性以前是 **validator** 属性。

view = "JSPFileName"

标识与 PageHandler 绑定的 Java Server Pages (JSP) 的名称和子目录路径。*JSPFileName* 是加引号的字符串。

缺省值是 PageHandler 的名称，并且文件扩展名为 **.jsp**。如果指定此属性，则包括文件扩展名（如果有的话）。

当保存或生成 PageHandler 时，EGL 会将一个 JSP 文件添加到项目中以便以后进行定制，除非在适当的文件夹（WebContent\WEB-INF 文件夹）中已存在同名的 JSP 文件（名称是在 **view** 属性中指定的）。EGL 决不会覆盖 JSP。

PageHandler 字段属性

PageHandler 字段属性指定在 PageHandler 部件中声明字段时有意义的特征。

属性如下所示：

- 第 628 页的 『 action 』
- 第 629 页的 『 byPassValidation 』
- 第 635 页的 『 displayName 』
- 第 635 页的 『 displayUse 』
- 第 637 页的 『 help 』
- 第 645 页的 『 newWindow 』
- 第 646 页的 『 numElementsItem 』
- 第 649 页的 『 selectFromListItem 』
- 第 649 页的 『 selectType 』

- 第 655 页的『 validationOrder 』
- 第 659 页的『 value 』

相关概念

- 第 59 页的『 EGL 属性概述 』
- 第 177 页的『 PageHandler 』

相关任务

- 第 175 页的『 创建 EGL pageHandler 部件 』
- 第 184 页的『 对 PageHandler 代码使用“快速编辑”视图 』

相关参考

- 第 622 页的『 PageHandler 部件属性 』
- 第 619 页的『 EGL 源格式的 PageHandler 部件 』
- 第 175 页的『 EGL 的 Page Designer 支持 』

pfKeyEquate

当声明引用文本表单的表单组时，属性 *pfKeyEquate* 指定当用户按下大编号功能键（PF13 至 PF24）时注册的击键是否与用户按下编号小于 12 的功能键时注册的击键相同。

如果对 *pfKeyEquate* 接受缺省值 *yes*，则逻辑表达式只能够引用 12 个功能键，这是因为（例如）PF2 与 PF14 相同。

注：PC 键盘上的功能键通常是 *F* 键，如 F1，但 EGL 使用 IBM *PF* 术语，所以 F1（例如）被称为 PF1。

相关概念

- 第 141 页的『 FormGroup 部件 』

相关参考

- 第 464 页的『 EGL 源格式的 FormGroup 部件 』

基本字段级别属性

下表列出 EGL 中的基本字段级别属性：

property	描述
action	标识当用户单击按钮或链接时调用的代码。
align	指定当数据长度比字段长度小时数据在变量字段中的位置。
byPassValidation	标识当用户单击按钮或链接时是否绕过基于 EGL 的验证。
color	指定文本表单中的字段的颜色。
column	指的是与项相关联的数据库表列的名称。缺省值是项的名称。

property	描述
currency	指示是否将值前面的货币符号包括到数字字段中，符号的精确位置由 zeroFormat 属性确定。
currencySymbol	指示在属性 currency 生效时要使用哪个货币符号。
dateFormat	标识日期的格式。
	指定当通过光笔或（对于仿真器会话）通过光标单击选择字段时是否设置该字段的已修正数据标记。
displayName	指定显示在字段旁边的标签。
displayUse	将 EGL 字段与用户界面控件相关联。
fieldLen	指定可以在文本表单字段中显示的单字节字符数。
fill	指示是否要求用户在每个字段位置中输入数据。
fillCharacter	指示用什么字符来填充文本或打印表单中或 PageHandler 数据中的未使用位置。
help	指定当用户将光标放到输入字段上方时将显示的悬浮式帮助文本。
highlight	指定用来显示字段的特殊效果（如果有的话）。
inputRequired	指示是否要求用户在字段中填写数据。
inputRequiredMsgKey	标识一条消息，如果字段属性 inputRequired 设置为 <i>yes</i> 并且用户未能在字段中填写数据时，将显示该消息。
intensity	指定显示字体的强度。
isBoolean	指示该字段表示布尔值。
isDecimalDigit	确定是否检查输入值是否仅包含十进制位
isHexDigit	确定是否检查输入值是否仅包含十六进制位
isNullable	表示是否可以将项设置为 NULL，如果与项相关联的表列可以被设置为 NULL，则此属性适用。
isReadOnly	表示是否应该从写到数据库或包含 FOR UPDATE OF 子句的缺省 SQL 语句中省略项及相关列。
lineWrap	指示文本在必要时是否换行以避免截断文本。
lowerCase	指示在用户的单字节字符输入中是否将字母字符设置为小写字母。
masked	指示是否显示用户输入的字符。
maxLen	指定写至数据库列的字段文本的最大长度。
minimumInput	指示要求用户在字段中填写的最小字符数（如果用户在字段中填写了任何数据的话）。

property	描述
minimumInputMsgKey	标识一条消息，如果用户执行下列操作，则将显示该消息： <ul style="list-style-type: none"> 在字段中填写数据；并且 填写的字符数小于属性 minimumInputRequired 中指定的值。
modified	指示程序是否认为字段已被修改，而无论用户是否更改了值。
needsSOSI	指示当用户在 ASCII 设备上输入类型为 MBCHAR 的数据时，EGL 是否执行特殊检查。
newWindow	指示当 EGL 运行时显示 Web 页面以便响应 action 属性中标识的活动时，是否使用新的浏览器窗口。
numElementsItem	标识一个 PageHandler 字段，该字段的运行时值指定要显示的数组元素的数目。
numericSeparator	指示是否在整数部分超过 3 位的数字中放置字符。
outline	允许在支持双字节字符的任何设备上在字段边缘画线。
pattern	将用户输入的文本与指定模式相匹配以便进行验证。
persistent	指示字段是否包括在为 SQL 记录生成的隐式 SQL 语句中。
protect	指定用户是否可以访问该字段。
selectFromListItem	标识用户可从中选择值的数组或 DataTable 列，这些值会传送至要声明的数组或基本字段。
selectType	指示检索到要声明的数组或基本字段中的值的种类。
sign	指示当用户输入或程序将数字放入字段时正号 (+) 或负号 (-) 的显示位置。
sqlDataCode	标识与记录项相关联的 SQL 数据类型。
sqlVariableLen	指示在 EGL 运行时将数据写至 SQL 数据库之前是否截断字符字段中的结尾空格和 NULL。
timeFormat	标识时间的格式。
timeStampFormat	标识在表单上显示的或在 PageHandler 中维护的时间戳记的格式。
typeChkMsgKey	标识一条消息，如果输入数据不适合于字段类型，则将显示该消息：
upperCase	指示在用户的单字节字符输入中是否将字母字符设置为大写字母。
validationOrder	指示该字段的验证器函数相对于任何其它字段的验证器函数的运行顺序。
validatorDataTable	标识一个验证器表，后者是一个 dataTable 部件，它是与用户输入进行比较的基础。

property	描述
validatorDataTableMsgKey	标识一个消息，如果用户提供的数据不符合验证器表（这是属性 validatorDataTable 中指定的表）的要求，则显示该信息。
validatorFunction	标识一个验证器函数，该函数是在 EGL 运行时执行基本验证检查（如果执行任何基本验证检查的话）之后运行的逻辑。
validatorFunctionMsgKey	标识显示的消息
validValues	指定对用户输入有效的一组值。
validValuesMsgKey	标识一条消息，如果设置了字段属性 validValues 并且用户在字段中填写了超出范围的数据，则将显示该消息。
value	标识显示 Web 页面时作为字段内容显示的字符串文字。
zeroFormat	指定在数字字段中（不是在类型为 MONEY 的字段中）如何显示零值。

action

当 EGL 属性 **displayUse** 为 *button* 或 *hyperlink* 时，属性 **action** 标识在用户单击按钮或链接时调用的代码。当您将该字段（或包括该字段的记录）放在 Page Designer 的“Web 页面”中时，赋给 **action** 的值将被用作缺省值。

action 的值为下列字符串文字中的其中一种：

- PageHandler 中的事件处理函数的名称
- 一个标签，它映射至 Web 资源（例如，映射至 JSP）并与 JSF 应用程序配置资源文件中的导航规则条目的源 - 结果属性相对应
- Java bean 中的方法的名称，在这种情况下，下列规则适用：
 - 格式为 bean 名称，后跟句点和方法名
 - 该 bean 名称必须与 JSF 应用程序配置资源文件中的其中一个受管 Bean 名称条目相关

如果未对 **action** 指定值，则用户对字段的单击将具有下列作用：

- 如果属性 **displayUse** 的值是 *button*，则会进行验证，然后 JSF 重新显示同一个 Web 页面。
- 如果属性 **displayUse** 的值是 *hyperlink*，则不进行验证，但 JSF 重新显示同一个 Web 页面。

相关概念

第 59 页的『EGL 属性概述』

第 177 页的『PageHandler』

相关任务

第 183 页的『将 JavaServer Faces 命令组件与 EGL PageHandler 绑定』

第 175 页的『创建 EGL pageHandler 部件』

第 184 页的『对 PageHandler 代码使用“快速编辑”视图』

相关参考

第 624 页的『PageHandler 字段属性』

第 622 页的『PageHandler 部件属性』

第 619 页的『EGL 源格式的 PageHandler 部件』

第 175 页的『EGL 的 Page Designer 支持』

align

align 属性指定当数据长度比字段长度小时数据在变量字段中的位置。

下列值属于枚举 **alignKind**:

left

将数据放在字段左边，对于字符数据，这是缺省情况。位于开头的空格将被除去，并且被放在字段末尾。

none

不对齐数据。此设置仅对字符数据有效。

right

将数据放在字段右边，对于数字数据，这是缺省情况。位于结尾的空格将被除去，并且被放在字段开头。对于带有小数位或正负号的数字数据，此设置是必需的。

该属性在 `DataItem` 部件中可用，并且对出现在下列上下文中的字段有意义:

- 控制台表单
- 打印表单
- 文本表单
- Web 页面

在输出时，字符和数字数据受此属性影响。对于输入，字符数据受此属性影响，但数字数据总是右对齐的。

相关概念

第 441 页的『EGL 中的枚举』

第 59 页的『EGL 属性概述』

相关参考

第 61 页的『定义格式的属性』

byPassValidation

当 EGL 属性 **displayUse** 为 `button` 或 `hyperlink` 时，属性 **byPassValidation** 标识在用户单击按钮或链接时是否绕过基于 EGL 的验证。您可能想要绕过验证以获取更好的性能；例如，每当用户单击“退出”按钮时。

当您将该字段（或包括该字段的记录）放在 Page Designer 的“Web 页面”中时，赋给 **byPassValidation** 的值将被用作缺省值。

该属性只影响基于 EGL 的验证，而不会影响 JSF 标记指定的验证；有关详细信息，请参阅 *PageHandler*。

下列值属于枚举布尔值:

no (缺省值)

像平常一样验证输入字段

yes

EGL 运行时不将用户数据返回至 PageHandler

相关概念

第 441 页的『EGL 中的枚举』

第 59 页的『EGL 属性概述』

第 177 页的『PageHandler』

相关任务

第 183 页的『将 JavaServer Faces 命令组件与 EGL PageHandler 绑定』

第 175 页的『创建 EGL pageHandler 部件』

第 184 页的『对 PageHandler 代码使用“快速编辑”视图』

相关参考

第 624 页的『PageHandler 字段属性』

第 622 页的『PageHandler 部件属性』

第 619 页的『EGL 源格式的 PageHandler 部件』

第 175 页的『EGL 的 Page Designer 支持』

color

color 属性指定文本表单中的字段的颜色。可以选择下列任何一项:

- black
- blue
- cyan
- defaultColor (缺省值)
- green
- magenta
- red
- white
- yellow

如果指定值 *defaultColor*, 则其它条件确定显示的颜色, 如下表所示。

是否表单中的所有字段都被赋予 <i>defaultColor</i> 值?	<i>protect</i> 的值	<i>intensity</i> 的值	对于被赋予 <i>defaultColor</i> 值的字段显示的颜色
yes	<i>yes</i> 或 <i>skip</i>	<i>not bold</i>	blue
yes	<i>yes</i> 或 <i>skip</i>	<i>bold</i>	white
yes	<i>no</i>	<i>not bold</i>	green
yes	<i>no</i>	<i>bold</i>	red
no	任何值	<i>not bold</i>	green
no	任何值	<i>bold</i>	white

相关概念

第 441 页的『EGL 中的枚举』

第 59 页的『EGL 属性概述』

相关参考

第 60 页的『字段显示属性』

column

属性 **column** 是指与项相关联的数据库表列的名称。缺省值是项的名称。列和相关项影响缺省 SQL 语句，如 SQL 支持中所述。

对于“*columnName*”，请替换为加引号的字符串、具有字符类型的变量或并置，如以下示例所示：

```
column = "Column" + "01"
```

如果列名是下列其中一个 SQL 保留字，则应该使用特殊的语法：

- CALL
- COLUMNS
- FROM
- GROUP
- HAVING
- INSERT
- ORDER
- SELECT
- SET
- UPDATE
- VALUES
- WHERE

如下示例所示，那些名称中的每一个都必须嵌入在两对引号中，并且每个内部引号的前面都必须要有转义字符（\）：

```
column = "\"SELECT\""
```

（如果将那些保留字用作表名，则会发生类似的情况。）

相关概念

第 400 页的『与 VisualAge Generator 的兼容性』

第 125 页的『记录类型和属性』

第 209 页的『SQL 支持』

第 23 页的『固定结构』

第 25 页的『Typedef』

相关任务

第 231 页的『检索 SQL 表数据』

相关参考

第 60 页的『字段显示属性』

第 511 页的『 add 』
第 517 页的『 close 』
第 430 页的『 数据初始化 』
第 520 页的『 delete 』
第 522 页的『 execute 』
第 533 页的『 get 』
第 544 页的『 get next 』
第 561 页的『 open 』
第 574 页的『 prepare 』
第 31 页的『 基本类型 』
第 673 页的『 记录和文件类型交叉引用 』
第 576 页的『 replace 』
第 579 页的『 set 』
第 680 页的『 SQL 数据代码和 EGL 主变量 』
第 860 页的『 terminalID 』
第 368 页的『 VAGCompatibility 』

currency

currency 属性指示是否将值前面的货币符号包括到数字字段中，符号的精确位置由 **zeroFormat** 属性确定。类型为 **MONEY** 的字段的格式取决于 **strLib.defaultMoneyFormat** 的值，它不受 **currency** 属性的影响。

currency 属性的值如下所示:

No (缺省值)

不使用货币符号。

Yes

使用 **currencySymbol** 中指定的符号。如果该处没有指定值，则使用缺省货币符号。

缺省货币符号由机器的语言环境确定。

该属性在 **DataItem** 部件中可用，并且对出现在下列上下文中的字段有意义:

- 打印表单
- 文本表单
- Web 页面

此属性用于输入和输出。

相关概念

第 441 页的『 EGL 中的枚举 』
第 59 页的『 EGL 属性概述 』

相关参考

第 61 页的『 定义格式的属性 』

currencySymbol

currencySymbol 属性指示在属性 **currency** 生效时要使用哪个货币符号。该值是字符串文字。

该属性在 `DataItem` 部件中可用，并且对出现在下列上下文中的字段有意义：

- 打印表单
- 文本表单
- Web 页面

此属性用于输入和输出。

相关概念

第 441 页的『EGL 中的枚举』

第 59 页的『EGL 属性概述』

相关参考

第 61 页的『定义格式的属性』

dateFormat

dateFormat 属性标识日期格式。

有效值如下所示：

"pattern"

pattern 值由一组字符组成，如日期、时间和时间戳记格式说明符中所述。

可从完整日期说明的开头或结尾删除字符，但不能从该说明的中间删除字符。

defaultDateFormat

如果对页面字段指定此选项，则 **defaultDateFormat** 的值是运行时 Java 语言环境中指定的日期格式。如果对表单字段指定此选项，则缺省模式等同于选择 **systemGregorianCalendar**。

eurDateFormat

模式“dd.MM.yyyy”，这是 IBM 欧洲标准日期格式。

isoDateFormat

模式“yyyy-MM-dd”，这是国际标准化组织（ISO）指定的日期格式。

jisDateFormat

模式“yyyy-MM-dd”，这是日本工业标准日期格式。

usaDateFormat

模式“MM/dd/yyyy”，这是 IBM 美国标准日期格式。

systemGregorianCalendar

8 个或 10 个字符的模式，它包括 dd（表示数字格式的天）、MM（表示数字格式的月份）和 yy 或 yyyy（表示数字格式的年份），并将 d、M、y 或数字以外的字符用作分隔符。

格式在以下 Java 运行时属性中：

```
vgj.datemask.gregorian.long.NLS
```

NLS

Java 运行时属性 **vgj.nls.code** 中指定的（本地语言支持）代码。此代码是 `targetNLS` 中列示的代码之一。不支持大写英语（代码 ENP）

有关 **vgj.nls.code** 的详细信息，请参阅 *Java 运行时属性（详细信息）*。

systemJulianDateFormat

6 个或 8 个字符的模式，它包含 DDD（表示数字格式的天）和 yy 或 yyyy（表示数字格式的年份），并将 D、y 或数字以外的字符用作分隔符。

格式在以下 Java 运行时属性中：

```
vgj.datemask.julian.long.NLS
```

NLS

Java 运行时属性 **vgj.nls.code** 中指定的（本地语言支持）代码。此代码是 targetNLS 中列示的代码之一。不支持大写英语（代码 ENP）

有关 **vgj.nls.code** 的详细信息，请参阅 *Java 运行时属性（详细信息）*。

该属性在 DataItem 部件中可用，并且对出现在下列上下文中的字段有意义：

- 控制台表单
- 打印表单
- 文本表单
- Web 页面

此属性用于输入和输出，但在下列情况下不使用此属性：

- 字段具有小数位、货币符号、数字分隔符或正负号；或者
- 字段的类型为 DBCHAR、MBCHAR 或 HEX；或者
- 字段不够长，无法包含反映掩码的值。有关其它详细信息，请参阅『日期的长度注意事项』。

内部日期格式

当用户输入有效数据时，日期将从对此字段指定的格式转换为内部格式，以便用于后续验证。

字符日期的内部格式与系统缺省格式相同，并且包含分隔符。

对于数字日期，内部格式如下所示：

- 对于格里历短日期，内部格式为 00yyMMdd
- 对于格里历长日期，内部格式为 00yyyyMMdd
- 对于儒略历短日期，内部格式为 0yyDDD
- 对于儒略历长日期，内部格式为 0yyyyDDD

日期的长度注意事项

在表单中，表单的字段长度必须与您指定的字段掩码的长度相匹配。该字段的长度必须足以容纳日期的内部格式。

在页面字段中，规则如下所示：

- 字段长度必须足以容纳指定的日期掩码，但可以更长
- 对于数字字段，长度计算不考虑分隔符。

下表提供了示例。

格式类型	示例	表单字段的长度	页面字段（字符类型）的最小长度	页面字段（数字类型）的有效长度
短格里历	yy/MM/dd	8	8	6
长格里历	yyyy/MM/dd	10	10	8
短儒略历	DDD-yy	6	6	5
长儒略历	DDD-yyyy	8	8	7

日期的 I/O 注意事项

将对输入到变量字段中的数据进行检查，以确保日期是按指定格式输入的。用户不需要对日期和月份输入前导零，而是可以指定 8/5/1996（示例）而不是 08/05/1996。然而，省略分隔符的用户必须输入全部前导零。

相关概念

第 315 页的『Java 运行时属性』

第 59 页的『EGL 属性概述』

相关参考

第 42 页的『日期、时间和时间戳记格式说明符』

第 61 页的『定义格式的属性』

第 493 页的『Java 运行时属性（详细信息）』

displayName

displayName 属性指定显示在字段旁边的标签。当您将该字段（或包括该字段的记录）放在 Page Designer 的“Web 页面”中时，所赋的值将被用作缺省值。

此属性的值为字符串文字。

相关概念

第 59 页的『EGL 属性概述』

第 177 页的『PageHandler』

相关任务

第 182 页的『将 EGL 记录与 Faces JSP 相关联』

第 175 页的『创建 EGL pageHandler 部件』

第 181 页的『创建 EGL 字段并将其与 Faces JSP 相关联』

第 184 页的『对 PageHandler 代码使用“快速编辑”视图』

相关参考

第 624 页的『PageHandler 字段属性』

第 622 页的『PageHandler 部件属性』

第 619 页的『EGL 源格式的 PageHandler 部件』

第 175 页的『EGL 的 Page Designer 支持』

displayUse

displayUse 属性将 EGL 字段与用户界面控件相关联。当您将该字段（或包括该字段的记录）放在 Page Designer 的“Web 页面”中时，所赋的值将被用作缺省值。

下列值属于枚举 **displayUseKind**:

button

控件具有按钮命令标记

secret

用户看不到该数据。此值适用于密码。

hyperlink

如果 **action** 属性是事件处理函数的名称，控件具有超链接命令标记。如果 **action** 属性为标签，则控件具有链接标记。在任一情况下，当用户单击该链接时，不进行验证，也不返回输入数据。

input

控件接受用户输入。一开始控件会显示 PageHandler 提供的值。

table

数据在表标记内。

output

PageHandler 字段输出（如果有的话）在控件中可视。

相关概念

第 441 页的『EGL 中的枚举』

第 59 页的『EGL 属性概述』

第 177 页的『PageHandler』

相关任务

第 182 页的『将 EGL 记录与 Faces JSP 相关联』

第 183 页的『将 JavaServer Faces 命令组件与 EGL PageHandler 绑定』

第 175 页的『创建 EGL pageHandler 部件』

第 181 页的『创建 EGL 字段并将其与 Faces JSP 相关联』

第 184 页的『对 PageHandler 代码使用“快速编辑”视图』

相关参考

第 624 页的『PageHandler 字段属性』

第 622 页的『PageHandler 部件属性』

第 619 页的『EGL 源格式的 PageHandler 部件』

第 175 页的『EGL 的 Page Designer 支持』

fieldLen

属性 **fieldLen** 指定可以在文本表单字段中显示的单字节字符数。此值不包括前导属性字节。

数字字段的 **fieldLen** 值的大小必须能够显示字段可容纳的最大数字以及小数点（如果该数字具有小数位的话）。CHAR、DBCHAR、MBCHAR 或 UNICODE 类型的字段的 **fieldLen** 值的大小必须能够处理双字节字符以及任何 shift-in/shift-out 字符。

fieldLen 的缺省值是显示基本类型（包括所有格式字符）所需的最大字节数。

相关概念

第 59 页的『EGL 属性概述』

相关参考

第 467 页的『EGL 源格式的表单部件』

fill

fill 属性表示是否要求用户在每个字段位置中输入数据。有效值为 *no* (缺省值) 和 *yes*。

相关概念

第 145 页的『文本表单』

相关参考

第 62 页的『验证属性』

第 722 页的『validationFailed()』

第 432 页的『EGL 源格式的 DataTable 部件』

第 836 页的『verifyChkDigitMod10()』

第 837 页的『verifyChkDigitMod11()』

fillCharacter

fillCharacter 属性指示用什么字符来填充文本或打印表单中的或者 PageHandler 数据中的未使用位置。另外，此属性更改了 *set field full* 的作用，如 *set* 所述。此属性仅影响输出。

对于数字，缺省值是空格，对于十六进制项，缺省值是 0。字符类型的缺省值取决于介质：

- 在文本或打印表单中，缺省值为空字符串
- 对于 PageHandler 数据，对于类型为 CHAR 或 MBCHAR 类型的数据，缺省值为空白

在 PageHandler 中，对于类型为 DBCHAR 或 UNICODE 的项，**fillCharacter** 的值必须是空格（这是缺省值）。

help

help 属性指定当用户将光标放到输入字段上方时将显示的悬浮式帮助文本。当您为 EGL 字段（或包括 EGL 字段的记录）放在 Page Designer 的“Web 页面”中时，所赋的值将被用作缺省值。

此属性的值为字符串文字。

相关概念

第 59 页的『EGL 属性概述』

第 177 页的『PageHandler』

相关任务

第 182 页的『将 EGL 记录与 Faces JSP 相关联』

第 175 页的『创建 EGL pageHandler 部件』

第 181 页的『创建 EGL 字段并将其与 Faces JSP 相关联』

第 184 页的『对 PageHandler 代码使用“快速编辑”视图』

相关参考

第 624 页的『PageHandler 字段属性』

第 622 页的『PageHandler 部件属性』

第 619 页的『EGL 源格式的 PageHandler 部件』

第 175 页的『EGL 的 Page Designer 支持』

highlight

highlight 属性指定用来显示字段的特殊效果（如果有的话）。有效值如下所示：

noHighLight（缺省值）

指示没有特殊效果；明确地说，没有闪烁、反转或下划线。

underline

在字段底部放置一条下划线。

相关概念

第 441 页的『EGL 中的枚举』

第 59 页的『EGL 属性概述』

相关参考

第 60 页的『字段显示属性』

inputRequired

inputRequired 属性表示是否要求用户在字段中填写数据。有效值为 *no*（缺省值）和 *yes*。

当属性值为 *yes* 时，如果用户未在字段中填写任何数据，则 EGL 运行时将显示一条消息，如有关字段属性 **inputRequiredMsgKey** 的内容所述。

相关概念

第 145 页的『文本表单』

相关参考

第 62 页的『验证属性』

第 722 页的『validationFailed()』

第 432 页的『EGL 源格式的 DataTable 部件』

第 836 页的『verifyChkDigitMod10()』

第 837 页的『verifyChkDigitMod11()』

inputRequiredMsgKey

属性 **inputRequiredMsgKey** 标识一条消息，如果字段属性 **inputRequired** 设置为 *yes* 并且用户未能在字段中填写数据时，将显示该消息。

消息表（包含消息的数据表）是在程序属性 **msgTablePrefix** 中标识的。有关数据表名的详细信息，请参阅 *EGL 源格式的 DataTable 部件*。

inputRequiredMsgKey 的值是一个字符串或文字，它与消息表中第一列的条目相匹配。

如果将数字键与期望字符键的消息表配合使用，则将把该数字转换为字符串。如果将字符串文字与期望数字键的消息表配合使用，则字符串中的值必须是带符号或无符号整数。

相关概念

第 145 页的『文本表单』

相关参考

第 62 页的『验证属性』

第 722 页的『validationFailed()』

第 432 页的『EGL 源格式的 DataTable 部件』

第 836 页的『verifyChkDigitMod10()』

第 837 页的『verifyChkDigitMod11()』

intensity

intensity 属性指定显示的字体的强度。有效值如下所示:

normalIntensity (缺省值)

将字段设置为可视, 并且没有粗体字效果。

bold

使文本以粗体字显示。

dim

当输入字段被禁用时, 适当地让文本以较低强度出现。

invisible

除去任何有关“字段位于表单中”的指示。

相关概念

第 441 页的『EGL 中的枚举』

第 59 页的『EGL 属性概述』

相关参考

第 60 页的『字段显示属性』

isBoolean

isBoolean 属性 (以前是 **boolean** 属性) 指示该字段表示布尔值。此属性对有效字段值作了限制, 在文本和打印表单中以及在 **PageHandler** 中, 此属性对输入或输出都很有用。

在与 EGL **PageHandler** 相关联的 Web 页面上, 布尔项是由复选框表示的。在表单上, 情况如下所示:

- 数字字段的值是 0 (表示 **false**) 或 1 (表示 **true**)。
- 字符字段的值是由从属于本地语言的字或部分字表示的。例如, 在英语中, 长度为三个或更多个字符的布尔字段具有值 **yes** (表示 **true**) 或 **no** (表示 **false**), 一字符布尔字段值具有截断的值 **y** 或 **n**。

yes 或 **no** 的特定字符值由语言环境确定。

isDecimalDigit

isDecimalDigit 属性确定是否检查输入值是否仅包含十进制位, 十进制位如下所示:

0123456789

有效值为 **no** (缺省值) 和 **yes**。

此属性仅适用于字符字段。

相关概念

第 145 页的『文本表单』

相关参考

第 62 页的『验证属性』

第 722 页的『validationFailed()』

第 432 页的『EGL 源格式的 DataTable 部件』

第 836 页的『verifyChkDigitMod10()』

第 837 页的『verifyChkDigitMod11()』

isHexDigit

isHexDigit 属性确定是否检查输入值是否仅包含十六进制位，十六进制位如下所示：

```
0123456789abcdefABCDEF
```

有效值为 *no*（缺省值）和 *yes*。

此属性仅适用于字符字段。

相关概念

第 145 页的『文本表单』

相关参考

第 62 页的『验证属性』

第 722 页的『validationFailed()』

第 432 页的『EGL 源格式的 DataTable 部件』

第 836 页的『verifyChkDigitMod10()』

第 837 页的『verifyChkDigitMod11()』

isNullable

属性 **isNullable** 指示是否可以将项设置为 NULL，如果与项相关联的表列可以被设置为 NULL，则此属性适用。有效值为 *yes*（缺省值）和 *no*。

对于 SQL 记录中的给定项，仅当 **isNullable** 设置为 *yes* 时，下列功能才可用：

- 程序可以将来自数据库的空值接受到项中。
- 程序可以使用 **set** 语句将项设置为 NULL，如 *set* 所述。该作用也是初始化项，如 *数据初始化* 中所述。
- 程序可以使用 **if** 语句来测试是否已将项设置为 NULL。

相关概念

第 400 页的『与 VisualAge Generator 的兼容性』

第 125 页的『记录类型和属性』

第 209 页的『SQL 支持』

第 23 页的『固定结构』

第 25 页的『Typedef』

相关任务

第 231 页的『检索 SQL 表数据』

相关参考

- 第 60 页的『字段显示属性』
- 第 511 页的『add』
- 第 517 页的『close』
- 第 430 页的『数据初始化』
- 第 520 页的『delete』
- 第 522 页的『execute』
- 第 533 页的『get』
- 第 544 页的『get next』
- 第 561 页的『open』
- 第 574 页的『prepare』
- 第 31 页的『基本类型』
- 第 673 页的『记录和文件类型交叉引用』
- 第 576 页的『replace』
- 第 579 页的『set』
- 第 680 页的『SQL 数据代码和 EGL 主变量』
- 第 860 页的『terminalID』
- 第 368 页的『VAGCompatibility』

isReadOnly

属性 **isReadOnly** 表示是否应该从写到数据库或包含 FOR UPDATE OF 子句的缺省 SQL 语句中省略项及相关列。缺省值为 *no*；但是在下列情况下，EGL 将结构项看作“只读”的：

- SQL 记录的属性 **key** 指示与结构项相关联的列是键列；或者
- SQL 记录部件与多个表相关联；或者
- SQL 列名是表达式。

相关概念

- 第 400 页的『与 VisualAge Generator 的兼容性』
- 第 125 页的『记录类型和属性』
- 第 209 页的『SQL 支持』
- 第 23 页的『固定结构』
- 第 25 页的『Typedef』

相关任务

- 第 231 页的『检索 SQL 表数据』

相关参考

- 第 60 页的『字段显示属性』
- 第 511 页的『add』
- 第 517 页的『close』
- 第 430 页的『数据初始化』
- 第 520 页的『delete』
- 第 522 页的『execute』
- 第 533 页的『get』
- 第 544 页的『get next』
- 第 561 页的『open』
- 第 574 页的『prepare』

- 第 31 页的『基本类型』
- 第 673 页的『记录和文件类型交叉引用』
- 第 576 页的『replace』
- 第 579 页的『set』
- 第 680 页的『SQL 数据代码和 EGL 主变量』
- 第 860 页的『terminalID』
- 第 368 页的『VAGCompatibility』

lineWrap

EGL 属性 **lineWrap** 指示文本在必要时是否自动换行以避免截断文本。

枚举 **lineWrapType** 的有效值如下：

character (缺省值)

字段中的文本不会在空格处断开。

compress

类型为 `ConsoleField` 的字段中的文本将在空格处断开，但当用户离开该字段时（浏览至另一字段或按 `ESC` 键），将除去用来使文本换行的所有多余空格。此值仅在控制台字段中有效。

word

如果可能，字段中的文本将在空格处断开。

属性 **lineWrap** 在 `DataItem` 部件中可用，并且对出现在下列上下文中的字段有意义：

- 控制台表单
- 打印表单
- 文本表单
- Web 页面

此属性会影响输入和输出。

相关概念

- 第 441 页的『EGL 中的枚举』
- 第 59 页的『EGL 属性概述』

相关参考

- 第 61 页的『定义格式的属性』

lowerCase

lowerCase 属性指示在用户的单字节字符输入中是否将字母字符设置为小写字母。值如下所示：

no (缺省值)

不要将用户的输入设置为小写。

yes

将用户的输入设置为小写。

masked

masked 属性指示是否显示用户输入的字符。此属性用于输入密码。值如下所示：

no (缺省值)

将显示用户输入的字符。

yes

将不显示用户输入的字符。

maxLen

属性 **maxLen** 指定写至数据库列的字段文本的最大长度。只要可能，此属性的缺省值将是该字段的长度；但如果字段的类型为 **STRING**，则不存在任何缺省值。

相关概念

第 400 页的『与 VisualAge Generator 的兼容性』

第 125 页的『记录类型和属性』

第 209 页的『SQL 支持』

第 23 页的『固定结构』

第 25 页的『Typedef』

相关任务

第 231 页的『检索 SQL 表数据』

相关参考

第 60 页的『字段显示属性』

第 511 页的『add』

第 517 页的『close』

第 430 页的『数据初始化』

第 520 页的『delete』

第 522 页的『execute』

第 533 页的『get』

第 544 页的『get next』

第 561 页的『open』

第 574 页的『prepare』

第 31 页的『基本类型』

第 673 页的『记录和文件类型交叉引用』

第 576 页的『replace』

第 579 页的『set』

第 680 页的『SQL 数据代码和 EGL 主变量』

第 860 页的『terminalID』

第 368 页的『VAGCompatibility』

minimumInput

minimumInput 属性指示要求用户在字段中填写的最小字符数（如果用户在字段中填写了任何数据的话）。缺省值是 0。

如果用户填写的字符数少于最小字符数，则 EGL 运行时将显示一条消息，如有关字段属性 **minimumInputMsgKey** 的内容所述。

相关概念

第 145 页的『文本表单』

相关参考

- 第 62 页的『验证属性』
- 第 722 页的『validationFailed()』
- 第 432 页的『EGL 源格式的 DataTable 部件』
- 第 836 页的『verifyChkDigitMod10()』
- 第 837 页的『verifyChkDigitMod11()』

minimumInputMsgKey

属性 **minimumInputMsgKey** 标识一条消息，如果用户执行下列操作，则将显示该消息：

- 在字段中填写数据；并且
- 填写的字符数小于属性 **minimumInputRequired** 中指定的值。

消息表（包含消息的表）是在程序属性 **msgTablePrefix** 中标识的。有关表名的详细信息，请参阅 *EGL 源格式的 DataTable 部件*。

minimumInputMsgKey 的值是一个字符串或文字，它与消息表中第一列的条目相匹配。

如果将数字键与期望字符键的消息表配合使用，则将把该数字转换为字符串。如果将字符串文字与期望数字键的消息表配合使用，则字符串中的值必须是带符号或无符号整数。

相关概念

第 145 页的『文本表单』

相关参考

- 第 62 页的『验证属性』
- 第 722 页的『validationFailed()』
- 第 432 页的『EGL 源格式的 DataTable 部件』
- 第 836 页的『verifyChkDigitMod10()』
- 第 837 页的『verifyChkDigitMod11()』

modified

指示程序是否认为字段已被修改，而无论用户是否更改了值。有关详细信息，请参阅 *已修正数据标记和 modified 属性*。

缺省值为 *no*。

相关概念

- 第 148 页的『已修正数据标记和 modified 属性』
- 第 59 页的『EGL 属性概述』

相关参考

第 467 页的『EGL 源格式的表单部件』

needsSOSI

needsSOSI 属性仅用于多字节字段（类型为 **MBCHAR** 的字段），并指示当用户在 ASCII 设备上输入类型为 **MBCHAR** 的数据时，EGL 是否执行特殊检查。有效值为 *yes*（缺省值）和 *no*。此项检查确定是否可以正确地将输入转换为主机 **SO/SI** 格式。

此属性是很有用的，这是因为在转换期间将删除多字节字符串末尾的结尾空格，以允许在每个双字节字符串两旁插入 SO/SI 定界符。为了正确地进行转换，对于多字节值中的每个双字节字符串，表单字段都必须至少有两个空格。

如果将 **needsSOSI** 设置为 *no*，则用户可以填写输入字段，在这种情况下，转换将截断数据而不发出警告。

然而，如果将 **needsSOSI** 设置为 *yes*，则当用户输入多字节数据时，结果如下：

- 由于提供了足够的空格，所以值被接受；或者
- 值被截断，并且用户接收到警告消息。

如果可能在 z/OS 或 iSeries 系统上使用用户在 ASCII 设备上输入的多字节数据，则将 **needsSOSI** 设置为 *yes*。

相关概念

第 145 页的『文本表单』

相关参考

第 62 页的『验证属性』

第 722 页的『validationFailed()』

第 432 页的『EGL 源格式的 DataTable 部件』

第 836 页的『verifyChkDigitMod10()』

第 837 页的『verifyChkDigitMod11()』

newWindow

属性 **newWindow** 指示当 EGL 运行时显示 Web 页面以便响应 **action** 属性中标识的活动时，是否使用新的浏览器窗口。

下列值属于枚举布尔值：

No (缺省值)

使用当前浏览器窗口来显示页面。

Yes

使用新的浏览器窗口。

如果未指定 **action** 属性，将使用当前浏览器窗口来显示页面。

相关概念

第 441 页的『EGL 中的枚举』

第 59 页的『EGL 属性概述』

第 177 页的『PageHandler』

相关任务

第 182 页的『将 EGL 记录与 Faces JSP 相关联』

第 183 页的『将 JavaServer Faces 命令组件与 EGL PageHandler 绑定』

第 175 页的『创建 EGL pageHandler 部件』

第 181 页的『创建 EGL 字段并将其与 Faces JSP 相关联』

第 184 页的『对 PageHandler 代码使用“快速编辑”视图』

相关参考

- 第 628 页的『action』
- 第 624 页的『PageHandler 字段属性』
- 第 622 页的『PageHandler 部件属性』
- 第 619 页的『EGL 源格式的 PageHandler 部件』
- 第 175 页的『EGL 的 Page Designer 支持』

numElementsItem

在结构字段数组上进行设置时，属性 **numElementsItem** 标识一个 PageHandler 字段，该字段的运行时值指定要显示的数组元素的数量。该属性只能用于输出，并且仅当在 occurs 值大于 1 的固定记录结构字段上进行设置时才有意义。

numElementsItem 的值是标识 PageHandler 字段的名称的字符串文字。因为动态数组包含指示正在使用的元素数量的指示符，所以该属性对动态数组无效；有关详细信息，请参阅数组。

相关概念

- 第 123 页的『固定记录部件』
- 第 59 页的『EGL 属性概述』
- 第 177 页的『PageHandler』

相关任务

- 第 182 页的『将 EGL 记录与 Faces JSP 相关联』
- 第 175 页的『创建 EGL pageHandler 部件』
- 第 181 页的『创建 EGL 字段并将其与 Faces JSP 相关联』
- 第 184 页的『对 PageHandler 代码使用“快速编辑”视图』

相关参考

- 第 68 页的『数组』
- 第 624 页的『PageHandler 字段属性』
- 第 622 页的『PageHandler 部件属性』
- 第 619 页的『EGL 源格式的 PageHandler 部件』
- 第 175 页的『EGL 的 Page Designer 支持』

numericSeparator

numericSeparator 属性表示是否在整数部分超过 3 位的数字中放置字符。例如，如果数字分隔符是逗号，则一千显示为 *1,000*，一百万显示为 *1,000,000*。值如下所示：

no (缺省值)

不使用数字分隔符。

yes

使用数字分隔符。

缺省值由机器的语言环境确定。

outline

outline 属性允许在支持双字节字符的任何设备上在字段边缘画线。有效值如下所示：

box

画线，以便在字段内容周围创建一个框

noOutline (缺省值)

不画线

另外，可以指定一个框的任何组件或所有组件。在这种情况下，将一个或多个值放在一对方括号中，用逗号将相邻的两个值隔开，如以下示例所示：

```
outline = [left, over, right, under]
```

部分值如下所示：

left

在字段的左边绘制一条垂直线

over

在字段的顶边绘制一条水平线

right

在字段的右边绘制一条垂直线

under

在字段的底边绘制一条水平线

每个表单字段的内容前面都有属性字节。您应该了解，不能在表单的最后一列中放置属性字节并希望 **outline** 值出现在下一列中，这将超出表单的边缘。（字段不会回绕到下一行。）同样，不能在表单的第一列中放置属性字节并期望 **outline** 值出现在该列中；**outline** 值仅可以出现在下一列中。

相关概念

第 441 页的『EGL 中的枚举』

第 59 页的『EGL 属性概述』

相关参考

第 60 页的『字段显示属性』

pattern

将用户输入的文本与指定模式相匹配以便进行验证。

相关概念

第 59 页的『EGL 属性概述』

相关参考

第 467 页的『EGL 源格式的表单部件』

persistent

属性 **persistent** 指示字段是否包括在为 SQL 记录生成的隐式 SQL 语句中。如果值为 *yes*，则在以下情况下，运行时将发生错误：

- 代码依赖于隐式 SQL 语句；并且
- 没有列与特定于字段的 **column** 属性的值相匹配。（缺省值为字段名。）

如果该变量在该数据库中没有对应的列，并且您想要将临时程序变量与 SQL 行相关联，则将 **persistent** 设置为 *no*。例如，您可能想要让变量指示程序是否修改了该行。

有关隐式 SQL 语句的详细信息，请参阅 *SQL 支持*。

相关概念

- 第 400 页的『与 VisualAge Generator 的兼容性』
- 第 125 页的『记录类型和属性』
- 第 209 页的『SQL 支持』
- 第 23 页的『固定结构』
- 第 25 页的『Typedef』

相关任务

- 第 231 页的『检索 SQL 表数据』

相关参考

- 第 60 页的『字段显示属性』
- 第 511 页的『add』
- 第 517 页的『close』
- 第 430 页的『数据初始化』
- 第 520 页的『delete』
- 第 522 页的『execute』
- 第 533 页的『get』
- 第 544 页的『get next』
- 第 561 页的『open』
- 第 574 页的『prepare』
- 第 31 页的『基本类型』
- 第 673 页的『记录和文件类型交叉引用』
- 第 576 页的『replace』
- 第 579 页的『set』
- 第 680 页的『SQL 数据代码和 EGL 主变量』
- 第 860 页的『terminalID』
- 第 368 页的『VAGCompatibility』

protect

指定用户是否可以访问该字段。有效值如下所示:

no (变量字段的缺省值)

设置字段, 使用户可以覆盖其中的值。

skip (常量字段的缺省值)

设置字段, 使用户不能覆盖其中的值。另外, 在下列任何一种情况下, 光标将跳过该字段:

- 用户正在依据跳进顺序在上一个字段中输入, 并且按下 **Tab** 键或者在上一个字段填充了内容; 或者
- 用户正在依据跳进顺序在下一个字段中输入, 并且按下 **Shift Tab** 键。

yes

设置字段, 使用户不能覆盖其中的值。

相关概念

- 第 59 页的『EGL 属性概述』

相关参考

- 第 467 页的『EGL 源格式的表单部件』

selectFromListItem

属性 **selectFromListItem** 标识用户可从中选择值的数组或 DataTable 列，这些值会传送到要声明的数组或基本字段。当您将该数组或基本字段放在 Page Designer 的“Web 页面”中时，指定给 **selectFromListItem** 的值将被用作缺省值。

属性 **selectFromListItem** 的值是标识源数组或 DataTable 列的字符串文字。

如果在声明数组时指定此属性，则允许用户选择多个值。如果在声明基本字段时指定此属性，则用户只能选择一个值。

从用户那里接收到的任何值必须与下列其中一种类型相对应：

- 用户选择的数组元素或 DataTable 列的内容；或者
- 数组或 DataTable 下标，它是一个整数，用来标识选择的元素或列。下标范围是从 1 到可用的元素数目。

属性 **selectType** 指示要接收的值的类型，它可以是用户选择的内容，也可以是数组的下标或列的索引。

相关概念

第 134 页的『DataTable』

第 59 页的『EGL 属性概述』

第 177 页的『PageHandler』

相关任务

第 182 页的『将 EGL 记录与 Faces JSP 相关联』

第 175 页的『创建 EGL pageHandler 部件』

第 181 页的『创建 EGL 字段并将其与 Faces JSP 相关联』

第 184 页的『对 PageHandler 代码使用“快速编辑”视图』

相关参考

第 68 页的『数组』

第 624 页的『PageHandler 字段属性』

第 622 页的『PageHandler 部件属性』

第 619 页的『EGL 源格式的 PageHandler 部件』

第 175 页的『EGL 的 Page Designer 支持』

『selectType』

selectType

属性 **selectType** 指示检索到要声明的数组或基本字段中的值的种类。当您将该数组或基本字段放在 Page Designer 的“Web 页面”中时，指定的值将被用作缺省值。

该值属于枚举 **selectTypeKind**：

index (缺省值)

要声明的数组或基本字段将接收响应用户选择的索引。在这种情况下，该数组或基本字段必须具有数字类型。

value

要声明的数组或基本字段将接收用户选择的值。在这种情况下，该项可以是任何类型。

有关背景知识信息，请参阅属性 **selectFromListItem**。

相关概念

第 134 页的『DataTable』

第 59 页的『EGL 属性概述』

第 177 页的『PageHandler』

相关任务

第 182 页的『将 EGL 记录与 Faces JSP 相关联』

第 175 页的『创建 EGL pageHandler 部件』

第 181 页的『创建 EGL 字段并将其与 Faces JSP 相关联』

第 184 页的『对 PageHandler 代码使用“快速编辑”视图』

相关参考

第 68 页的『数组』

第 624 页的『PageHandler 字段属性』

第 622 页的『PageHandler 部件属性』

第 619 页的『EGL 源格式的 PageHandler 部件』

第 175 页的『EGL 的 Page Designer 支持』

第 649 页的『selectFromListItem』

sign

sign 属性指示当用户输入或程序将数字放入字段时正号 (+) 或负号 (-) 的显示位置。值如下所示：

none

不显示符号。

leading

缺省值：符号显示在数字第一位的左边，符号的准确位置由 **zeroFormat** 属性（在后文中有所描述）确定。

trailing

在数字最后一位的右边紧跟着显示符号。

sqlDataCode

属性 **sqlDataCode** 的值是一个数字，它标识与记录项相关联的 SQL 数据类型。如果在声明时、验证时或生成的程序运行时访问数据库管理系统，则该系统将使用数据代码。

仅当为 VisualAge Generator 兼容性设置了环境时，属性 **sqlDataCode** 才可用。有关详细信息，请参阅与 *VisualAge Generator* 的兼容性。

缺省值取决于记录项的基本类型和长度，如下表所示。有关其它详细信息，请参阅 *SQL 数据代码*。

EGL 基本类型	长度	SQL 数据代码
BIN	4	501
	9	497

EGL 基本类型	长度	SQL 数据代码
CHAR	<=254	453
	>254 和 <=4000	449
	>4000	457
DBCHAR	<=127	469
	>127 和 <=2000	465
	>2000	473
DECIMAL	any	485
HEX	any	481
UNICODE	<=127	469
	>127 和 <=2000	465
	>2000	473

相关概念

- 第 400 页的『与 VisualAge Generator 的兼容性』
- 第 125 页的『记录类型和属性』
- 第 209 页的『SQL 支持』
- 第 23 页的『固定结构』
- 第 25 页的『Typedef』

相关任务

- 第 231 页的『检索 SQL 表数据』

相关参考

- 第 60 页的『字段显示属性』
- 第 511 页的『add』
- 第 517 页的『close』
- 第 430 页的『数据初始化』
- 第 520 页的『delete』
- 第 522 页的『execute』
- 第 533 页的『get』
- 第 544 页的『get next』
- 第 561 页的『open』
- 第 574 页的『prepare』
- 第 31 页的『基本类型』
- 第 673 页的『记录和文件类型交叉引用』
- 第 576 页的『replace』
- 第 579 页的『set』
- 第 680 页的『SQL 数据代码和 EGL 主变量』
- 第 860 页的『terminalID』
- 第 368 页的『VAGCompatibility』

sqlVariableLen

属性 **sqlVariableLen**（以前为 **sqlVar** 属性）的值指示在 EGL 运行时将数据写至 SQL 数据库之前是否截断字符字段中的结尾空格和 NULL。此属性对非字符数据不起作用。

如果对应的 SQL 表列具有 `varchar` 或 `vargraphic` SQL 数据类型，则指定 `yes`。

相关概念

第 400 页的『与 VisualAge Generator 的兼容性』

第 125 页的『记录类型和属性』

第 209 页的『SQL 支持』

第 23 页的『固定结构』

第 25 页的『Typedef』

相关任务

第 231 页的『检索 SQL 表数据』

相关参考

第 60 页的『字段显示属性』

第 511 页的『add』

第 517 页的『close』

第 430 页的『数据初始化』

第 520 页的『delete』

第 522 页的『execute』

第 533 页的『get』

第 544 页的『get next』

第 561 页的『open』

第 574 页的『prepare』

第 31 页的『基本类型』

第 673 页的『记录和文件类型交叉引用』

第 576 页的『replace』

第 579 页的『set』

第 680 页的『SQL 数据代码和 EGL 主变量』

第 860 页的『terminalID』

第 368 页的『VAGCompatibility』

timeFormat

timeFormat 属性标识时间格式。

有效值如下所示：

"pattern"

pattern 值由一组字符组成，如日期、时间和时间戳记格式说明符中所述。

可从完整时间说明的开头或结尾删除字符，但不能从该说明的中间删除字符。

defaultTimeFormat

Java 环境中的缺省值由 Java 语言环境设置。

eurTimeFormat

模式 *HH.mm.ss*，这是 IBM 欧洲标准时间格式。

isoTimeFormat

模式 *HH.mm.ss*，这是国际标准化组织（ISO）指定的时间格式。

jisTimeFormat

模式 *HH:mm:ss*，这是日本工业标准时间格式。

usaTimeFormat

模式 *hh:mm AM*, 这是 IBM 美国标准时间格式。

该属性在 `DataItem` 部件中可用, 并且对出现在下列上下文中的字段有意义:

- 控制台表单
- 打印表单
- 文本表单
- Web 页面

此属性用于输入和输出, 但在下列情况下不使用此属性:

- 字段具有小数位、货币符号、数字分隔符或正负号; 或者
- 字段的类型为 `DBCHAR`、`MBCHAR` 或 `HEX`; 或者
- 字段不够长, 无法包含反映掩码的值。有关其它详细信息, 请参阅[时间的长度注意事项](#)。

时间的长度注意事项

在表单中, 字段长度必须与您指定的时间掩码的长度相匹配。在页面字段中, 规则如下所示:

- 项长度必须足以容纳指定的时间掩码, 但可以更长
- 对于数字项, 长度计算不考虑分隔符。

时间的 I/O 注意事项

将对输入到变量字段中的数据进行检查, 以确保时间是按指定格式输入的。用户不需要对小时、分钟和秒输入前导零, 而是可以指定 `8:15` (示例) 而不是 `08:15`。然而, 省略分隔符的用户必须输入全部前导零。

以内部格式存储的时间不会被识别为时间, 而仅仅被识别为数据。例如, 如果将 6 字符时间字段移至长度为 10 的字符项中, 则 EGL 将用空格来填充目标字段。然而, 当在表单上显示 6 字符值时, 将把时间从其内部格式进行适当的转换。

相关概念

第 315 页的『[Java 运行时属性](#)』

第 59 页的『[EGL 属性概述](#)』

相关参考

第 42 页的『[日期、时间和时间戳记格式说明符](#)』

第 61 页的『[定义格式的属性能](#)』

第 493 页的『[Java 运行时属性 \(详细信息\)](#)』

timeStampFormat

`timeStampFormat` 属性标识在表单上显示的或在 `PageHandler` 中维护的时间戳记的格式。

有效值如下所示:

"pattern"

pattern 值由一组字符组成, 如[日期、时间和时间戳记格式说明符](#)中所述。

可从完整时间戳记说明的开头或结尾删除字符, 但不能从该说明的中间删除字符。

defaultTimeStampFormat

在 Java 环境中，缺省值由 Java 语言环境设置。

db2TimestampFormat

模式 `yyyy-MM-dd-HH.mm.ss.ffffff`，这是 IBM DB2 缺省时间戳记格式。

odbcTimestampFormat

模式 `yyyy-MM-dd HH:mm:ss.ffffff`，这是 ODBC 时间戳记格式。

相关概念

第 315 页的『Java 运行时属性』

第 59 页的『EGL 属性概述』

相关参考

第 42 页的『日期、时间和时间戳记格式说明符』

第 493 页的『Java 运行时属性（详细信息）』

typeChkMsgKey

属性 **typeChkMsgKey** 标识一条消息，如果输入数据不适合于字段类型，则将显示该消息：

消息表（包含消息的表）是在程序属性 **msgTablePrefix** 中标识的。有关表名的详细信息，请参阅 *EGL 源格式* 的 *DataTable* 部件。

typeChkMsgKey 的值是一个字符串或文字，它与消息表中第一列的条目相匹配。

如果将数字键与期望字符键的消息表配合使用，则将把该数字转换为字符串。如果将字符串文字与期望数字键的消息表配合使用，则字符串中的值必须是带符号或无符号整数。

相关概念

第 145 页的『文本表单』

相关参考

第 62 页的『验证属性』

第 722 页的『`validationFailed()`』

第 432 页的『EGL 源格式的 *DataTable* 部件』

第 836 页的『`verifyChkDigitMod10()`』

第 837 页的『`verifyChkDigitMod11()`』

upperCase

upperCase 属性指示在用户的单字节字符输入中是否将字母字符设置为大写字母。

此属性在表单中以及在 `PageHandler` 中都很有用。

upperCase 的值如下所示：

No（缺省值）

不要将用户的输入设置为大写。

Yes

将用户的输入设置为大写。

validationOrder

属性 **validationOrder** 指示相对于任何其它字段的验证器函数，该字段的验证器函数何时运行。如果一个字段的验证取决于另一个字段的先前验证，则此属性很重要。

该值是一个文字整数。

首先对任何被指定了 **validationOrder** 属性值的字段进行验证，并且，首先验证具有最小数值的项。然后，对任何未指定 **validationOrder** 值的项进行验证，在这种情况下，验证顺序是字段在 `PageHandler` 中的定义顺序。

相关概念

第 59 页的『EGL 属性概述』

第 177 页的『PageHandler』

相关任务

第 175 页的『创建 EGL pageHandler 部件』

第 181 页的『创建 EGL 字段并将其与 Faces JSP 相关联』

第 184 页的『对 PageHandler 代码使用“快速编辑”视图』

相关参考

第 624 页的『PageHandler 字段属性』

第 622 页的『PageHandler 部件属性』

第 619 页的『EGL 源格式的 PageHandler 部件』

第 175 页的『EGL 的 Page Designer 支持』

validatorDataTable

validatorDataTable 属性（以前是 **validatorTable** 属性）标识一个验证器表，后者是一个 `dataTable` 部件，它是与用户输入进行比较的基础。验证器表的使用发生在 EGL 运行时执行基本验证检查（如果执行任何基本验证检查的话）之后。有关下列属性的内容描述了那些基本检查：

- `inputRequired`
- `isDecimalDigit`
- `isHexDigit`
- `minimumInput`
- `needsSOSI`
- `validValues`

在使用 **validatorFunction** 属性之前进行全部的检验，该属性指定一个验证函数，该验证函数完成跨值验证。

可以指定具有下列任何一种类型的验证器表，如 EGL 格式的 `DataTable` 部件中所述：

matchInvalidTable

指示用户的输入必须与数据表第一列中的任何值不相同。

matchValidTable

指示用户的输入必须与数据表第一列中的值相匹配。

rangeChkTable

指示用户的输入必须与一个值相匹配，该值至少介于一个数据表行的第一列和第二列的值之间。（范围是包括边界的；当用户的输入与任何行的第一列或第二列的值相匹配时，输入也有效。）

如果验证失败，则显示的消息取决于属性 **validatorDataTableMsgKey** 的值。

相关概念

第 145 页的『文本表单』

相关参考

第 62 页的『验证属性』

第 722 页的『validationFailed()』

第 432 页的『EGL 源格式的 DataTable 部件』

第 836 页的『verifyChkDigitMod10()』

第 837 页的『verifyChkDigitMod11()』

validatorDataTableMsgKey

属性 **validatorDataTableMsgKey**（以前是 **validatorTableMsgKey** 属性）标识一个消息，如果用户提供的数据不符合验证器表（这是属性 **validatorDataTable** 中指定的表）的要求，则显示该信息。

消息表（包含消息的表）是在程序属性 **msgTablePrefix** 中标识的。有关消息表名的详细信息，请参阅 *EGL 源格式的 DataTable 部件*。

validatorDataTableMsgKey 的值是一个字符串或文字，它与消息表中第一列的条目相匹配。

如果将数字键与期望字符键的消息表配合使用，则将把该数字转换为字符串。如果将字符串文字与期望数字键的消息表配合使用，则字符串中的值必须是带符号或无符号整数。

相关概念

第 145 页的『文本表单』

相关参考

第 62 页的『验证属性』

第 722 页的『validationFailed()』

第 432 页的『EGL 源格式的 DataTable 部件』

第 836 页的『verifyChkDigitMod10()』

第 837 页的『verifyChkDigitMod11()』

validatorFunction

validatorFunction 属性（以前是 **validator** 属性）标识一个验证器函数，该函数是在 EGL 运行时执行基本验证检查（如果执行任何基本验证检查的话）之后运行的逻辑。有关下列属性的内容描述了那些检查：

- inputRequired
- isDecimalDigit
- isHexDigit

- `minimumInput`
- `needsSOSI`
- `validValues`

基本检查先于验证器表的使用（如有关 `validatorDataTable` 属性的内容所述），并且所有检查都先于 `validatorFunction` 属性的使用。由于验证器函数可以执行跨字段检查，并且这样的检查通常需要有效的字段值，所以，此事件顺序很重要。

`validatorFunction` 值是您编写的验证器函数。可以按以下方式编写该函数：不带参数，并且，如果该函数检测到错误，则它通过调用 `ConverseLib.validationFailed` 来请求重新显示表单。

如果在指定了两个系统函数的其中一个时验证失败，则显示的消息取决于属性 `validatorFunctionMsgKey` 的值。然而，如果当指定了您自己的验证器函数时验证失败，则函数不使用 `validatorFunctionMsgKey`，而是通过调用 `ConverseLib.validationFailed` 来显示消息。

相关概念

第 145 页的『文本表单』

相关参考

第 62 页的『验证属性』

第 722 页的『`validationFailed()`』

第 432 页的『EGL 源格式的 `DataTable` 部件』

第 836 页的『`verifyChkDigitMod10()`』

第 837 页的『`verifyChkDigitMod11()`』

validatorFunctionMsgKey

属性 `validatorFunctionMsgKey`（以前是 `validatorMsgKey` 属性）标识消息，该消息在以下情况下显示：

- 属性 `validatorFunction` 指示使用 `sysLib.verifyChkDigitMod10` 或 `sysLib.verifyChkDigitMod11`；并且
- 指定的函数指示用户的输入出错。

消息表（包含消息的表）是在程序属性 `msgTablePrefix` 中标识的。有关表名的详细信息，请参阅 *EGL 源格式的 `DataTable` 部件*。

`validatorFunctionMsgKey` 的值是一个字符串或文字，它与消息表中第一列的条目相匹配。

如果将数字键与期望字符键的消息表配合使用，则将把该数字转换为字符串。如果将字符串文字与期望数字键的消息表配合使用，则字符串中的值必须是带符号或无符号整数。

相关概念

第 145 页的『文本表单』

相关参考

第 62 页的『验证属性』

第 722 页的『`validationFailed()`』

第 432 页的『EGL 源格式的 DataTable 部件』

第 836 页的『verifyChkDigitMod10()』

第 837 页的『verifyChkDigitMod11()』

validValues

validValues 属性（以前是 **range** 属性）指示对用户输入有效的一组值。此属性用于数字或字符字段。此属性的格式如下所示：

```
validValues = arrayLiteral
```

arrayLiteral

单值和双值元素的数组文字，如以下示例中所示：

```
validValues = [ [1,3], 5, 12 ]  
validValues = [ "a", ["bbb", "i"]]
```

每个单值元素包含一个有效值。每个双值元素包含一个范围：

- 对于数字，最左边的值是最低的有效值，最右边的值是最高有效值。在前一示例中，值 1、2 和 3 对于类型为 INT 的字段是有效的。
- 对于字符字段，用户输入将与一定范围的值进行比较（只要可以对字符数进行比较）。例如，范围 ["a", "c"] 包括（有效）其首个字符为“a”、“b”或“c”的任何输入。尽管字符串“cat”在整理顺序中大于“c”，但“cat”是有效输入。

一般规则如下所示：如果范围中的第一个值被称为 *lowValue* 并且第二个值被称为 *highValue*，则用户的输入在满足下列任何测试时有效：

- 用户输入等于 *lowValue* 或 *highValue*
- 用户输入大于 *lowValue* 并小于 *highValue*
- 输入字符的首字符与 *lowValue* 中的字符的首字符相匹配（只要可以进行比较）
- 输入字符的首字符与 *highValue* 中的字符的首字符相匹配（只要可以进行比较）

其它示例如下所示：

```
// valid values are 1, 2, 3, 5, 7, 9, and 11  
validValues = [[1, 3], 5, 7, 11]  
  
// valid values are the letters "a" and "z"  
validValues = ["a", "z"]  
  
// valid values are any string beginning with "a"  
validValues = ["a", "a"]  
  
// valid values are any string  
// beginning with a lowercase letter  
validValues = ["a", "z"]
```

如果用户的输入超出指定的范围，则 EGL 运行时将显示一条消息，如有关字段属性 **validValuesMsgKey** 的内容所述。

相关概念

第 145 页的『文本表单』

相关参考

第 62 页的『验证属性』

第 722 页的『validationFailed()』

第 432 页的『EGL 源格式的 DataTable 部件』

第 836 页的『verifyChkDigitMod10()』

第 837 页的『verifyChkDigitMod11()』

validValuesMsgKey

属性 **validValuesMsgKey**（以前是 **rangeMsgKey** 属性）标识一条消息，如果设置了字段属性 **validValues** 并且用户在字段中填写了超出范围的数据，则将显示该消息。

消息表（包含消息的表）是在程序属性 **msgTablePrefix** 中标识的。有关表名的详细信息，请参阅 *EGL 源格式的 DataTable 部件*。

validValuesMsgKey 的值是一个字符串或文字，它与消息表中第一列的条目相匹配。

如果将数字键与期望字符键的消息表配合使用，则将把该数字转换为字符串。如果将字符串文字与期望数字键的消息表配合使用，则字符串中的值必须是带符号或无符号整数。

此属性仅适用于数字字段。

相关概念

第 145 页的『文本表单』

相关参考

第 432 页的『EGL 源格式的 DataTable 部件』

第 722 页的『validationFailed()』

第 62 页的『验证属性』

第 836 页的『verifyChkDigitMod10()』

第 837 页的『verifyChkDigitMod11()』

value

属性 **value** 标识显示 Web 页面时作为字段内容显示的字符串文字。当您将 EGL 字段放在 Page Designer 的“Web 页面”中时，该文字被用作缺省值。

相关概念

第 59 页的『EGL 属性概述』

第 177 页的『PageHandler』

相关任务

第 182 页的『将 EGL 记录与 Faces JSP 相关联』

第 181 页的『创建 EGL 字段并将其与 Faces JSP 相关联』

第 175 页的『创建 EGL pageHandler 部件』

第 184 页的『对 PageHandler 代码使用“快速编辑”视图』

相关参考

第 624 页的『PageHandler 字段属性』

第 622 页的『PageHandler 部件属性』

第 619 页的『EGL 源格式的 PageHandler 部件』

第 175 页的『EGL 的 Page Designer 支持』

zeroFormat

zeroFormat 属性指定在数字字段中（但在类型为 MONEY 的字段中）如何显示零值。此属性受 **numeric separator**、**currency** 和 **fillCharacter** 属性影响。**zeroFormat** 的值如下所示：

Yes

零值作为数字零显示，后者可以表示为具有小数点（一个示例为 0.00，如果项被定义为具有两个小数位的话）和具有货币符号和字符分隔符（\$000,000.00 是一个示例，这取决于 **currency** 和 **numericSeparator** 属性的值）。当 **zeroFormat** 属性的值为 *yes* 时，下列规则适用：

- 如果填充字符（**fillCharacter** 属性的值）为 0，则通过字符 0 定义数据的格式
- 如果填充字符为空，则数据是向左对齐的
- 如果填充字符是空格，则数据是向右对齐的
- 如果填充字符是星号（*），则星号取代空格用作左填充符

No

将零值显示成一系列填充字符。

相关参考

第 493 页的『Java 运行时属性（详细信息）』

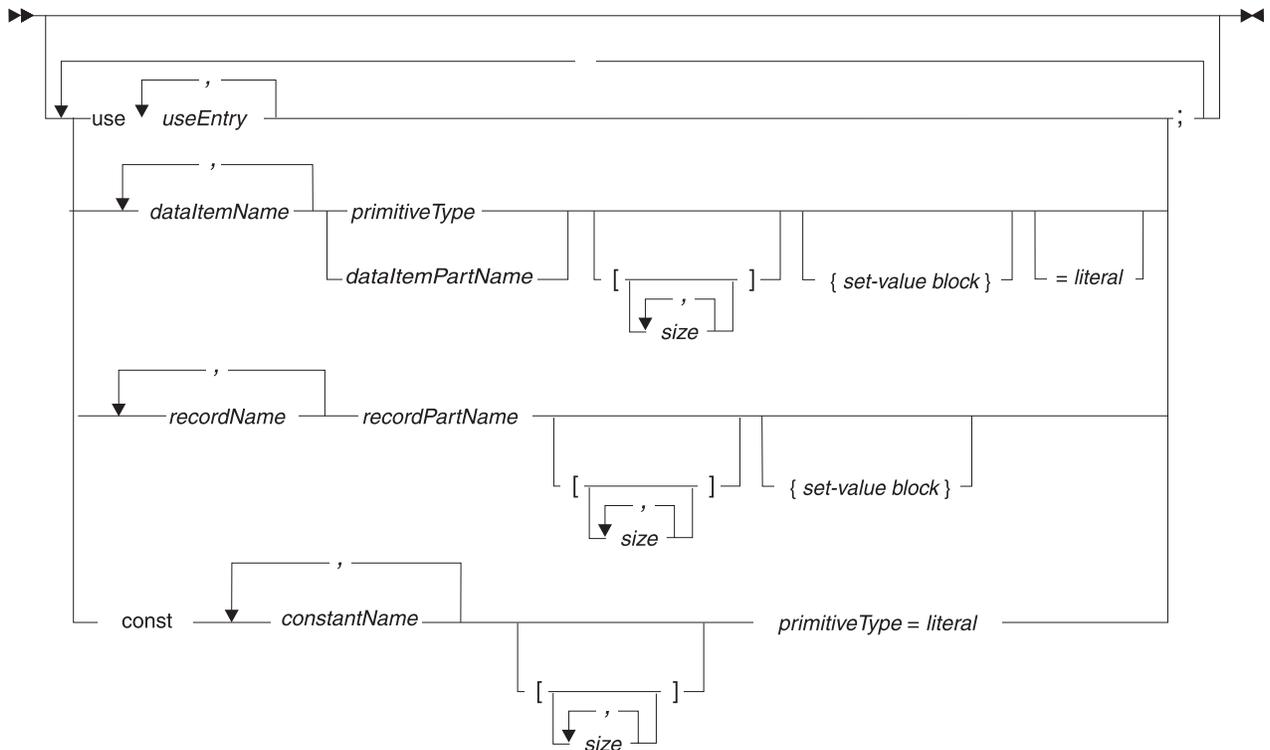
第 579 页的『set』

第 351 页的『currencySymbol』

第 42 页的『日期、时间和时间戳记格式说明符』

参数以外的程序数据

程序数据的语法图如下所示：



use useEntry

提供对 `dataTable` 或库的更简便的访问，并且是访问 `formGroup` 中的表单所需的。有关详细信息，请参阅使用声明。

dataItemName

基本字段的名称。有关命名规则，请参阅命名约定。

primitiveType

基本字段的类型或（对于数组）数组元素的基本类型。根据该类型，可能需要下列信息：

- 参数的长度或者（对于数组）数组元素的长度。长度是一个整数，它表示内存区中的字符或数字的数目。
- 对于某些数字类型，可以指定用来表示小数点后的位数的整数。小数点不与数据存储在一起。
- 对于类型为 `INTERVAL` 或 `TIMESTAMP` 的项，可指定日期时间掩码，它会赋予项值中的给定位置特别的意义（如“年份位”）。

有关详细信息，请参阅基本类型及有关给定类型的主题。

dataItemPartName

对程序可视的 `dataItem` 部件的名称。有关可视性的详细信息，请参阅对部件的引用。

该部件作为格式模型，如 *Typedef* 所述。

size

数组中的元素数目。如果指定元素数目，则数组是使用该数目的元素进行初始化的。

set-value block

有关详细信息，请参阅 *EGL 属性概述* 和 *Set-value 块*。

recordName

记录的名称。有关命名规则，请参阅 *命名约定*。

recordPartName

对程序可视的记录部件的名称。有关可视性的详细信息，请参阅 *对部件的引用*。

该部件作为格式模型，如 *Typedef* 所述。

const *constantName primitiveType=literal*

常量的名称、类型和值。指定用引号括起来的字符串（对于字符类型）、数字（对于数字类型）或一组相应类型的值（对于数组）。示例如下所示：

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

有关命名规则，请参阅 *命名约定*。

相关概念

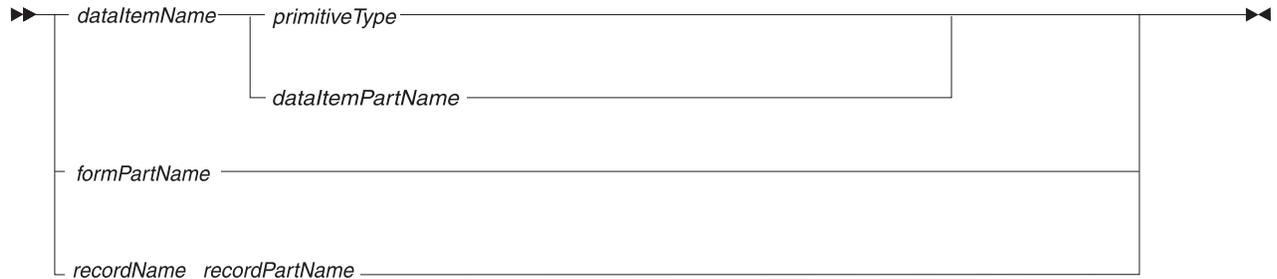
- 第 13 页的『*EGL 项目、包和文件*』
- 第 59 页的『*EGL 属性概述*』
- 第 16 页的『*部件*』
- 第 128 页的『*程序部件*』
- 第 53 页的『*引用 EGL 中的变量*』
- 第 147 页的『*文本应用程序中的分段*』
- 第 62 页的『*set-value 块*』
- 第 690 页的『*EGL 语句和命令的语法图*』
- 第 25 页的『*Typedef*』

相关参考

- 第 68 页的『*数组*』
- 第 430 页的『*数据初始化*』
- 第 431 页的『*EGL 源格式的 DataItem 部件*』
- 第 432 页的『*EGL 源格式的 DataTable 部件*』
- 第 448 页的『*EGL 源格式*』
- 第 80 页的『*EGL 语句*』
- 第 531 页的『*forward*』
- 第 481 页的『*EGL 源格式的函数部件*』
- 第 488 页的『*EGL 源格式的带索引记录部件*』
- 第 672 页的『*输入表单*』
- 第 672 页的『*输入记录*』
- 第 38 页的『*INTERVAL*』
- 第 490 页的『*I/O 错误值*』
- 第 603 页的『*EGL 源格式的 MQ 记录部件*』
- 第 612 页的『*命名约定*』
- 第 31 页的『*基本类型*』
- 第 676 页的『*EGL 源格式的相关记录部件*』
- 第 679 页的『*EGL 源格式的串行记录部件*』
- 第 683 页的『*EGL 源格式的 SQL 记录部件*』

程序参数

程序参数的语法图如下所示:



dataItemName

基本字段的名称。有关命名规则，请参阅命名约定。

primitiveType

基本字段的类型。根据该类型，可能需要下列信息:

- 参数的长度，它是一个整数，表示内存区中的字符或数字的数目。
- 对于某些数字类型，可以指定用来表示小数点后的位数的整数。小数点不与数据存储在一起。
- 对于类型为 INTERVAL 或 TIMESTAMP 的项，可指定日期时间掩码，它会赋予项值中的给定位置特别的意义（如“年份位”）。

dataItemPartName

对程序可视的 *dataItem* 部件的名称。有关可视性的详细信息，请参阅对部件的引用。

该部件作为格式模型，如 *Typedef* 所述。

formPartName

表单的名称。

必须能够通过程序的其中一个使用声明中标识的 *formGroup* 来访问该表单。不能将作为参数访问的表单显示给用户，但它可以提供对从另一个程序传递的字段值的访问。

有关命名规则，请参阅命名约定。

recordName

记录或固定记录的名称。有关命名规则，请参阅命名约定。

recordPartName

对程序可视的记录部件（或固定记录部件）的名称。有关可视性的详细信息，请参阅对部件的引用。

该部件作为格式模型，如 *Typedef* 所述。

下列语句适用于对记录参数进行的输入或输出（I/O）:

- 从另一程序传递的记录不包括记录状态，如 I/O 错误值 *endOfFile*。同样，不会将记录状态的任何更改返回给调用程序，所以，如果对记录参数执行 I/O，则对该记录执行的任何测试都必须在程序结束前进行。
- 对记录执行的任何 I/O 操作都使用对参数指定的记录属性而不是对自变量指定的记录属性。
- 对于 *indexedRecord*、*mqRecord*、*relativeRecord* 或 *serialRecord* 类型的记录，与记录声明相关联的文件或消息队列被视为运行单元资源而不是程序资源。每当记录属性 **fileName**（或 **queueName**）具有相同的值时，本地记录声明就共享同一个文件（或队列）。无论在运行单元中有多少个记录与文件或队列相关联，每次都仅有一个物理文件可以与文件或队列名相关联，并且，EGL 在适当情况下可以通过关闭并重新打开文件来确保此规则的实施。

从另一个 EGL 程序发送的自变量在引用时必须与相关参数相兼容。有关详细信息，请参阅 *EGL* 中的引用兼容性。

相关概念

- 第 128 页的『程序部件』
- 第 20 页的『对部件的引用』
- 第 53 页的『引用 EGL 中的变量』
- 第 690 页的『EGL 语句和命令的语法图』
- 第 25 页的『Typedef』

相关参考

- 第 68 页的『数组』
- 第 345 页的『EGL 源格式的基本记录部件』
- 第 431 页的『EGL 源格式的 DataItem 部件』
- 第 448 页的『EGL 源格式』
- 第 488 页的『EGL 源格式的带索引记录部件』
- 第 38 页的『INTERVAL』
- 第 612 页的『命名约定』
- 第 31 页的『基本类型』
- 第 675 页的『EGL 中的引用兼容性』
- 第 676 页的『EGL 源格式的相关记录部件』
- 第 679 页的『EGL 源格式的串行记录部件』
- 第 683 页的『EGL 源格式的 SQL 记录部件』
- 第 40 页的『TIMESTAMP』

EGL 源格式的程序部件

可以在 EGL 文件中声明程序部件，EGL 源格式对该部件作了描述。当您编写该文件时，执行下列操作：

- 仅包括那些由程序独占使用的部件
- 不要包括其它主部件（数据表、库、程序或 *pageHandler*）

下一个示例显示了一个带有两个嵌入函数以及一个独立函数和一个独立记录部件的被调用程序部件：

```
Program myProgram type basicProgram (employeeNum INT)
{
    includeReferencedFunctions = yes
```

```

}

// program-global variables
employees record_ws;
employeeName char(20);

// a required embedded function
Function main()
  // initialize employee names
  recd_init();

  // get the correct employee name
  // based on the employeeNum passed
  employeeName = getEmployeeName(employeeNum);
end

// another embedded function
Function recd_init()
  employees.name[1] = "Employee 1";
  employees.name[2] = "Employee 2";
end

end

// stand-alone function
Function getEmployeeName(employeeNum INT) returns (CHAR(20))

  // local variable
  index BIN(4);
  index = 2;
  if (employeeNum > index)
    return("Error");
  else
    return(employees.name[employeeNum]);
  end

end

// record part that acts as a typeDef for employees
Record record_ws type basicRecord
  10 name CHAR(20)[2];
end

```

有关其它详细信息，请参阅特定类型的程序的主题。

相关概念

第 16 页的『部件』

第 128 页的『程序部件』

相关参考

第 666 页的『EGL 源格式的基本程序』

第 448 页的『EGL 源格式』

第 481 页的『EGL 源格式的函数部件』

第 668 页的『EGL 源格式的文本用户界面程序』

EGL 源格式的基本程序

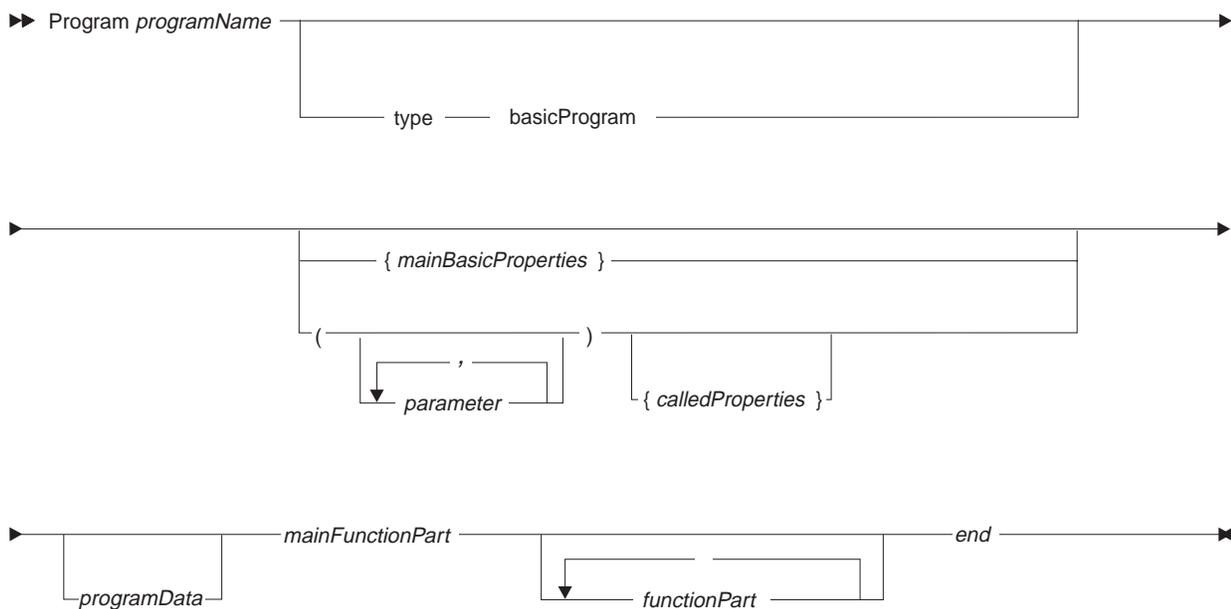
下面是基本程序的示例:

```
program 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
```

`basicProgram` 类型的程序部件的语法图如下所示:



Program *programPartName* ... end

将该部件标识为程序部件并指定名称和类型。如果程序名后面跟着左圆括号，则该程序是被调用基本程序。

如果未设置 **alias** 属性（如后文所述），则生成的程序的名称是 *programPartName*。

有关其它规则，请参阅 [命名约定](#)。

mainBasicProperties

主基本程序的属性是可选的:

- **alias**
- **allowUnqualifiedItemReferences**
- **handleHardIOErrors**
- **includeReferencedFunctions**
- **inputRecord**

- **localSQLScope**
- **msgTablePrefix**
- **throwNrfEofExceptions**

有关详细信息，请参阅程序属性。

parameter

指定参数名，该参数可以是数据项、记录或表单；也可以是记录动态数组或数据项动态数组。有关规则，请参阅命名约定。

如果调用程序的自变量是变量（不是常量或文字），则对参数进行的任何更改都将更改可供调用程序使用的内存区。

用逗号将每个参数与下一个参数隔开。有关其它详细信息，请参阅程序参数。

calledProperties

被调用属性是可选的：

- **alias**
- **allowUnqualifiedItemReferences**
- **handleHardIOErrors**
- **includeReferencedFunctions**
- **localSQLScope**
- **msgTablePrefix**
- **throwNrfEofExceptions**

有关详细信息，请参阅程序属性。

programData

变量和使用声明，如参数以外的程序数据中所述。

mainFunctionPart

名为 *main* 的必需函数，该函数不接受参数。（能接受参数的程序代码只有程序本身以及除 *main* 以外的函数。）

有关编写函数的详细信息，请参阅 *EGL* 源格式的函数部件。

functionPart

一个嵌入函数，它是此程序的专用函数。有关编写函数的详细信息，请参阅 *EGL* 源格式的函数部件。

相关概念

- 第 13 页的『EGL 项目、包和文件』
- 第 59 页的『EGL 属性概述』
- 第 16 页的『部件』
- 第 128 页的『程序部件』
- 第 690 页的『EGL 语句和命令的语法图』

相关参考

- 第 448 页的『EGL 源格式』
- 第 481 页的『EGL 源格式的函数部件』
- 第 612 页的『命名约定』
- 第 660 页的『参数以外的程序数据』
- 第 663 页的『程序参数』

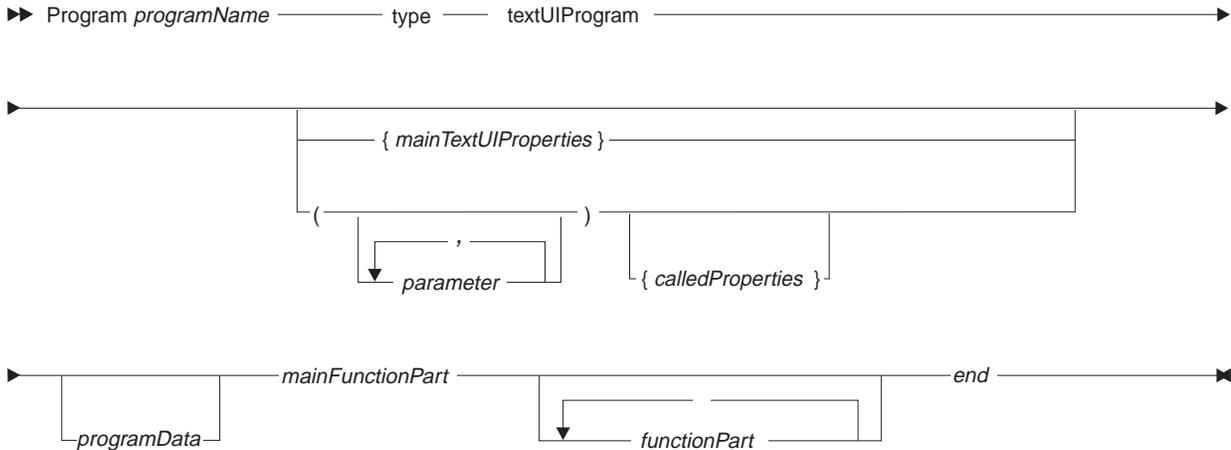
第 664 页的『EGL 源格式的程序部件』

第 670 页的『程序部件属性』

第 875 页的『使用声明』

EGL 源格式的文本用户界面程序

`textUIProgram` 类型的程序部件的语法图如下所示:



Program *programPartName* ... end

将该部件标识为程序部件并指定名称和类型。如果程序名后面跟着左圆括号，则该程序是被调用基本程序。

如果未设置 **alias** 属性（如后文所述），则生成的程序的名称是 *programPartName*。如果未设置 **alias** 属性（如后文所述），则生成的程序的名称是 *programPartName* 或者（如果要生成 COBOL 的话）*programPartName* 的前 8 个字符。

有关其它规则，请参阅命名约定。

mainTextUIProperties

主文本用户界面程序的属性是可选的:

- **alias**
- **allowUnqualifiedItemReferences**
- **handleHardIOErrors**
- **includeReferencedFunctions**
- **inputForm**
- **inputRecord**
- **localSQLScope**
- **msgTablePrefix**
- **segmented**
- **throwNrfEofExceptions**

有关详细信息，请参阅程序属性。

parameter

指定参数名，该参数可以是数据项、记录或表单；也可以是记录动态数组或数据项动态数组。有关规则，请参阅命名约定。

如果调用程序的自变量是变量（不是常量或文字），则对参数进行的任何更改都将更改可供调用程序使用的内存区。

用逗号将每个参数与下一个参数隔开。有关其它详细信息，请参阅程序参数。

calledProperties

被调用属性是可选的：

- **alias**
- **allowUnqualifiedItemReferences**
- **includeReferencedFunctions**
- **msgTablePrefix**

有关详细信息，请参阅程序属性。

programData

变量和使用声明，如参数以外的程序数据中所述。

mainFunctionPart

名为 *main* 的必需函数，该函数不接受参数。（能接受参数的程序代码只有程序本身以及除 *main* 以外的函数。）

有关编写函数的详细信息，请参阅 *EGL 源格式的函数部件*。

functionPart

嵌入的函数，该函数对于除程序以外的任何逻辑部件都不可用。有关编写函数的详细信息，请参阅 *EGL 源格式的函数部件*。

下面是文本用户界面程序的示例：

```
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
```

相关概念

第 13 页的『EGL 项目、包和文件』

第 59 页的『EGL 属性概述』

第 16 页的『部件』

第 128 页的『程序部件』

第 147 页的『文本应用程序中的分段』

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 448 页的『EGL 源格式』

第 481 页的『EGL 源格式的函数部件』
第 612 页的『命名约定』
第 660 页的『参数以外的程序数据』
第 663 页的『程序参数』
第 664 页的『EGL 源格式的程序部件』
『程序部件属性』
第 875 页的『使用声明』

程序部件属性

被调用程序和主程序具有不同的部件属性，即便同为主程序，基本类型和文本用户界面类型的主程序的部件属性也各不相同。属性如下所示：

alias = *"alias"*

一个字符串，它包含在生成的输出的名称中。如果未设置 **alias** 属性，则会使用程序部件名。

alias 属性在任何程序中都可使用。

allowUnqualifiedItemReferences = no, allowUnqualifiedItemReferences = yes

指定是否允许代码在引用结构中的项时省略容器和子结构限定符。

allowUnqualifiedItemReferences 属性在任何程序中都可使用。

例如，请参照以下记录部件：

```
Record aRecordPart type basicRecord
  10 myItem01 CHAR(5);
  10 myItem02 CHAR(5);
end
```

以下变量基于该部件：

```
myRecord aRecordPart;
```

如果接受 **allowUnqualifiedItemReferences** 的缺省值 (*no*)，则在引用 `myItem01` 时必须指定记录名，如以下赋值所示：

```
myValue = myRecord.myItem01;
```

然而，如果将属性 **allowUnqualifiedItemReferences** 设置为 *yes*，则无需指定记录名：

```
myValue = myItem01;
```

建议您接受缺省值，这样有助于实现最佳实践。通过指定容器名，可以减少对阅读代码的人以及对 EGL 造成的歧义。

EGL 使用一组规则来确定变量名或项名所引用的内存区。有关详细信息，请参阅对变量和常量的引用。

handleHardIOErrors = yes, handleHardIOErrors = no

设置系统变量 **VGVar.handleHardIOErrors** 的缺省值。该变量控制在 `try` 块中的 I/O 操作发生硬错误后程序是否继续运行。该属性的缺省值为 *yes*，它将该变量设置为 1。

从 VisualAge Generator 迁移的代码可能不会像以前一样工作，除非将 **handleHardIOErrors** 设置 *no*，这会将该变量设置为 0。

此属性在任何程序中可用。有关其它详细信息，请参阅 *VGVar.handleHardIOErrors* 和异常处理。

includeReferencedFunctions = no, includeReferencedFunctions = yes

指示程序是否包含每个既不在程序中也不在程序所访问的库中的函数的副本。

includeReferencedFunctions 属性在任何程序中都可使用。

缺省值为 *no*，这表示如果在开发时按照建议遵守以下惯例，则可以忽略此属性：

- 将共享函数放在库中
- 将非共享函数放在程序中

如果正在使用不在库中的共享函数，则仅当将属性 **includeReferencedFunctions** 设置为 *yes* 时才有可能进行生成。

inputForm = "formName"

标识在程序逻辑运行之前显示给用户的表单，如输入表单中所述。

inputForm 属性仅在主文本用户界面程序中可用。

inputRecord = "inputRecord"

标识程序自动初始化的全局基本记录，该记录可以从使用 **transfer** 语句来转移控制权的程序接收数据。有关其它详细信息，请参阅输入记录。

inputRecord 属性在任何主程序中都可使用。

localSQLScope = yes, localSQLScope = no

指示 SQL 结果集和预编译语句的标识对于程序是不是局部的（在缺省情况下是局部的）。如果接受值 *yes*，则表示不同程序可独立使用相同的标识。

如果指定 *no*，则标识在整个运行单元中共享。在当前代码中创建的标识在其它地方也是可用的，尽管其它代码可使用 **localSQLScope = yes** 来阻止访问这些标识。而且，当前代码可以引用在其它位置创建的标识，但仅当其它代码已经运行并且没有阻止访问时才会如此。

共享 SQL 标识的影响如下所示：

- 可以在一个程序中打开某个结果集并从另一个程序中获取该结果集中的行
- 可以在一个程序中预编译某个 SQL 语句并且在另一个程序中运行该语句

localSQLScope 属性在任何程序中都可使用。

msgTablePrefix = "prefix"

指定用作程序消息表的数据表的名称的第一个到第四个字符。名称中的其它字符与 EGL 源格式的 *DataTable* 部件中列示的其中一个本地语言代码相对应。

msgTablePrefix 属性在任何基本程序或文本用户界面程序中都可使用。

在 Web 应用程序中运行的程序不使用消息表，但使用 JavaServer Faces 消息资源。有关该资源的详细信息，请参阅下列主题中有关 **msgResource** 属性的描述：

- EGL 源格式的 *PageHandler* 部件

segmented = no, segmented = yes

指示程序是否是分段程序，分段对该类程序作了说明。在主文本用户界面程序中，缺省值为 *no*。在其它类型的程序中，该属性无效。

throwNrfEofExceptions = no, throwNrfEofExceptions = yes

指定软错误是否导致抛出异常。缺省值为 *no*。有关背景知识信息，请参阅异常处理。

相关概念

- 第 128 页的『程序部件』
- 第 53 页的『引用 EGL 中的变量』
- 第 147 页的『文本应用程序中的分段』

相关参考

- 第 432 页的『EGL 源格式的 DataTable 部件』
- 第 86 页的『异常处理』
- 第 531 页的『forward』
- 『输入表单』
- 『输入记录』
- 第 612 页的『命名约定』
- 第 619 页的『EGL 源格式的 PageHandler 部件』
- 第 690 页的『EGL 语句和命令的语法图』

第 867 页的『handleHardIOErrors』

输入表单

当声明在文本应用程序中运行的主程序时，可以选择指定输入表单，这是在程序逻辑运行之前显示给用户的表单。

有两种可能的方案：

- 如果该程序是 EGL 生成的程序中的 `show-form-returning-to` 语句的目标，则发送程序向用户显示表单，并且该表单必须与接收程序的输入表单完全相同。仅在用户提交表单之后才调用接收程序。在用户提交表单之后，接收程序不会再次显示输入表单，而是运行初始逻辑（执行函数）。
- 如果该程序是程序（EGL 程序或非 EGL 程序）中的 `transfer` 语句的目标，或者该程序是由用户或操作系统命令调用的，则接收程序转换输入表单。（在这种情况下，该表单上的输入字段在显示之前会被初始化。）在用户提交表单之后，初始逻辑（执行函数）运行。

输入表单必须位于您在程序部件声明中指定的表单组中。

相关参考

- 第 430 页的『数据初始化』

输入记录

任何主程序部件都可以有输入记录，这是 EGL 生成的程序自动初始化的全局记录。记录必须具有 `basicRecord` 类型。

如果程序是作为通过记录进行转移的结果启动的，则程序初始化输入记录（这是该程序的内部记录），然后将传送的数据赋给该记录。

如果输入记录的长度比接收到的数据长，则输入记录中的额外区域将保留记录初始化期间被赋予的值。如果输入记录的长度比接收到的数据短，则将截断额外的数据。

如果所传送的数据的基本类型与输入记录中相应位置中的基本类型不兼容，则接收程序可能会异常结束。

相关概念

第 59 页的『EGL 属性概述』

第 16 页的『部件』

第 400 页的『与 VisualAge Generator 的兼容性』

相关参考

第 430 页的『数据初始化』

记录和文件类型交叉引用

下表按目标平台显示了记录类型与文件类型之间的关联。

相关概念

第 125 页的『记录类型和属性』

第 277 页的『资源关联和文件类型』

相关任务

第 280 页的『将资源关联部件添加至 EGL 构建文件』

第 281 页的『编辑 EGL 构建文件中的资源关联部件』

第 282 页的『从 EGL 构建文件中除去资源关联部件』

相关参考

第 361 页的『resourceAssociations』

支持变长记录的属性

当声明记录部件时，可以包括支持使用变长记录的属性。可以将变长串行记录用于访问顺序文件，可以将变长串行记录或带索引记录用于访问 VSAM 文件，并且可以将变长 MQ 记录用于访问 MQSeries 消息队列。

具有 `lengthItem` 属性的变长记录

`lengthItem` 属性（如果存在的话）标识在下列情况下使用的项：

- 代码从文件或队列中读取记录。长度项接收读入变长记录中的字节数。
- 代码写记录。长度项指定要添加到文件或队列中的字节数。

长度项可以是下列任何一项：

- 同一记录中的结构项
- 记录中对于程序来说是全局的或对于访问该记录的函数来说是局部的结构项（可以使用在程序或函数中声明的记录变量来限定长度项）
- 对于程序来说是全局的或对于访问记录的函数来说是局部的数据项

长度项具有下列特征：

- 具有基本类型 BIN、DECIMAL、INT、NUM 或 SMALLINT
- 不包含小数位
- 最多允许 9 位数

以下是一个带有 `lengthItem` 属性的变长记录部件的示例:

```
Record mySerialRecordPart1 type serialRecord
{
  fileName = "myFile",
  lengthItem = "myOtherField"
}
10 myField01 BIN(4); // 2 bytes long
10 myField02 NUM(3); // 3 bytes long
10 myField03 CHAR(20); // 20 bytes long
end
```

除非项是字符项, 否则, 在写记录时, 长度项的值必须落在项边界之间。例如, 类型为 `mySerialRecordPart1` 的记录可以具有长度项 `myOtherField`, 该长度项设置为 2, 5, 6, 7, ... , 24, 25。 `myOtherField` 设置为 2 的记录仅包含 `myField01` 的值; `myOtherField` 设置为 5 的记录包含 `myField01` 和 `myField02` 的值; `myOtherField` 设置为 6 到 24 的记录还包含 `myField03` 的一部分。

具有 `numElementsItem` 属性的变长记录

`NumElementsItem` 属性 (如果存在的话) 标识当代码对文件或队列进行添加或更新操作时使用的项。变长记录必须将一个数组作为最后一个顶级结构项。元素数目项中的值表示写入的实际数组元素数目。该值的范围可以从 0 到最大值 (最大值是在记录中的最后一个顶级结构项的声明中指定的 *occurs* 值)。

写入的字节数等于下列各项之和:

- 记录的定长部分的字节数。
- 元素数目项的值乘以结束数组的每个元素中的字节数。

元素数目项具有下列特征:

- 具有基本类型 `BIN`、`DECIMAL`、`INT`、`NUM` 或 `SMALLINT`
- 不包含小数位
- 最多允许 9 位数

以下是一个带有 `numElementsItem` 属性的变长记录部件的示例:

```
Record mySerialRecordPart2 type serialRecord
{
  fileName = "myFile",
  numElementsItem = "myField02"
}
10 myField01 BIN(4); // 2 bytes long
10 myField02 NUM(3); // 3 bytes long
10 myField03 CHAR(20)[3]; // 60 bytes long
20 mySubField01 CHAR(10);
20 mySubField02 CHAR(10);
end
```

在元素数目项 `myField02` 设置为 2 的情况下写入类型为 `mySerialRecordPart2` 的记录导致将具有 `myField01`、`myField02` 以及 `myField03` 的两次出现的变长记录写入文件或队列。

元素数目项必须是变长记录的固定长度部分中的项。使用未限定引用来命名元素数目项。例如, 使用 `myField02` 而不是 `myRecord.myField02`。

当从文件中读取记录时, 元素数目项不起作用。

同时具有 lengthItem 和 numElementsItem 属性的变长记录

如果对变长记录同时指定了 lengthItem 和 numElementsItem 属性，则使用元素数目项来计算记录的长度。在将记录写入文件之前，将把计算出的长度移动到记录长度项中。

在进行调用或转移时传递的变长记录

如果在进行调用时传递了变长记录，则下列描述是适用的：

- 为该记录保留空间，并且空间的大小就是对该记录指定的最大长度
- 如果 callLink 元素的属性 **type** 的值为 remoteCall 或 ejbCall，则长度项（如果有的话）必须位于记录内部；有关详细信息，请参阅 *callLink* 元素

同样，如果在转移时传递了变长记录，则为该记录保留空间，并且空间的大小就是对该记录指定的最大长度。

相关概念

第 242 页的『MQSeries 支持』
第 125 页的『记录类型和属性』

相关参考

第 370 页的『callLink 元素』
第 606 页的『MQ 记录属性』

EGL 中的引用兼容性

参数或变量是一个内存区。在某些情况下，变量包含重要的业务数据；如特定名称或职员标识。在另外一些情况下，变量是引用变量；它包含用来在运行时访问业务数据的值（明确地说是内存地址）。

在将一个非引用变量赋给另一个非引用变量时，将得到两份同样的业务数据。例如，如果赋值语句中的源变量包含特定职员标识，则该语句会使目标变量也包含该标识。但是，将一个引用变量赋给另一个引用变量时，源和目标都将包含用于访问同一内存区的值。

引用兼容性规则（如后所述）在下列情况下适用：

- 将一个引用变量赋给另一个引用变量时；或者
- 当 EGL 在自变量与相关参数之间传送数据时，但仅当处于下列其中一种情况时才会这样：
 - 接收函数中的参数具有修饰符 INOUT。
 - 该参数在 PageHandler 的 onPageLoad 函数中。
 - 该参数在由另一 EGL 程序调用的 EGL 程序中。

在这些情况下，自变量为源，该参数为目标。

引用兼容性的规则如下所示：

- 一个引用变量只能赋给或传送给同一类型的另一个引用变量。
- 当源（或自变量）引用基本类型或一组 DataItem 时，下列陈述适用：

- 基本特征（如果有的话）必须完全相同。例如，类型为 CHAR(6) 的自变量与类型为 CHAR(7) 的参数不兼容。
- 可空自变量与可空或非可空参数是兼容的。不可空自变量仅与不可空参数是兼容的。
- 不同包中的部件被视为不同类型，但 DataItem 部件例外。
- 对于固定记录或结构字段，自变量的长度必须大于或等于参数的长度。此规则能够阻止接收代码访问无效内存。

相关概念

第 177 页的『PageHandler』

相关参考

第 477 页的『函数参数』

第 481 页的『EGL 源格式的函数部件』

第 619 页的『EGL 源格式的 PageHandler 部件』

第 663 页的『程序参数』

第 664 页的『EGL 源格式的程序部件』

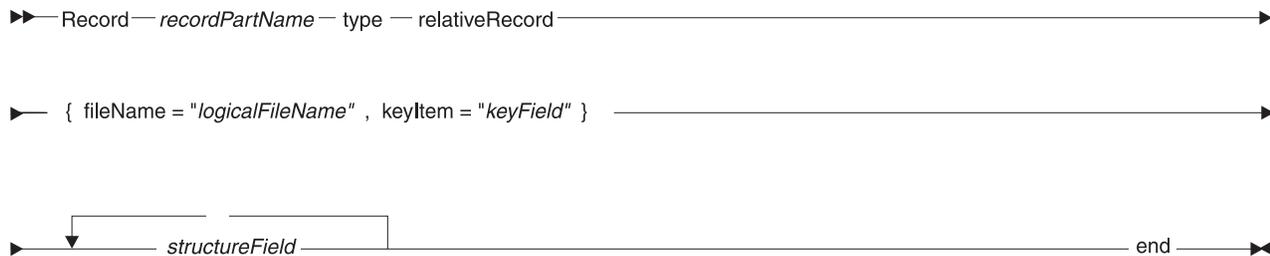
EGL 源格式的相关记录部件

在 EGL 文件中声明类型为 relativeRecord 的固定记录部件，EGL 源格式中对该部件作了描述。

下面是相关记录部件的示例：

```
Record myRelativeRecordPart type relativeRecord
{
  fileName = "myFile",
  keyItem = "myKeyItem"
}
10 myKeyItem NUM(4);
10 myContent CHAR(76);
end
```

相关记录部件的语法图如下所示：



Record recordPartName relativeRecord

标识具有 relativeRecord 类型的部件并指定名称。有关命名规则，请参阅命名约定。

fileName = "logicalFileName"

逻辑文件名。有关输入的含义的详细信息，请参阅资源关联（概述）。有关规则，请参阅命名约定。

keyItem = "keyField"

键字段，它可以是下列内存区中的任何一个：

- 同一固定记录中的字段
- 对于程序来说是全局的，对于访问该固定记录的函数来说是局部的变量或字段

必须使用未限定引用来命名该键字段。例如，使用 *myField* 而不是 *myRecord.myField*。（但是，在函数中，可以象引用任何字段一样引用键字段。）键字段在访问记录的函数的局部作用域中必须是唯一的，或者，必须是不在局部作用域中并在全局作用域中是唯一的。

键字段具有下列特征：

- 具有基本类型 NUM、BIN、DECIMAL、INT 或 SMALLINT
- 不包含小数位
- 最多允许 9 位数

只有 **get** 和 **add** 语句使用相对记录键字段，但是键字段必须可供任何使用固定记录来进行文件访问的函数使用。

structureField

结构字段，如 *EGL* 源格式的结构字段中所述。

相关概念

第 13 页的『EGL 项目、包和文件』

第 20 页的『对部件的引用』

第 16 页的『部件』

第 122 页的『记录部件』

第 53 页的『引用 EGL 中的变量』

第 25 页的『Typedef』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 68 页的『数组』

第 431 页的『EGL 源格式的 DataItem 部件』

第 448 页的『EGL 源格式』

第 481 页的『EGL 源格式的函数部件』

第 488 页的『EGL 源格式的带索引记录部件』

第 603 页的『EGL 源格式的 MQ 记录部件』

第 612 页的『命名约定』

第 31 页的『基本类型』

第 664 页的『EGL 源格式的程序部件』

第 277 页的『资源关联和文件类型』

第 679 页的『EGL 源格式的串行记录部件』

第 683 页的『EGL 源格式的 SQL 记录部件』

第 686 页的『EGL 源格式中的结构字段』

运行单元

运行单元是通过本地调用或（在某些情况下）通过转移而相关的一组程序。每个运行单元都具有下列特征：

- 各程序作为一个组一起运作。当发生硬错误而未进行处理时，将从内存中除去运行单元中的所有程序。
- 各程序共享相同的运行时属性。例如，相同的数据库和文件在整个运行单元中都可用，并且，当调用 `sysLib.connect` 或 `VGLib.connectionService` 来动态地连接至数据库时，该连接存在于同一个运行单元中的任何接收到控制权的程序中。

Java 运行单元由单一线程中运行的程序组成。新的运行单元可以在用户调用程序时随主程序一起启动。**transfer** 语句也调用主程序，但继续使用同一个运行单元。

在下列情况下，被调用程序是运行单元的初始程序：

- 该调用是来自 EJB 会话 bean 的调用；或者
- 该调用是远程调用，但在以下情况中，继续使用同一个运行单元：
 - 被调用程序是由 EGL 或 VisualAge Generator 生成的；并且
 - 调用不涉及 TCP/IP 侦听器。

Java 运行单元中的所有程序都受到相同 Java 运行时属性的影响。

相关概念

第 315 页的『Java 运行时属性』

第 282 页的『链接选项部件』

相关参考

第 230 页的『缺省数据库』

第 820 页的『connect()』

第 839 页的『connectionService()』

resultSetID

结果集标识具有 EGL 语法，当您正在访问关系数据库并需要使下列类型的语句相关时，使用此标识：

- 首先，选择结果集的 **open** 或 **get** 语句，或者调用返回结果集的存储过程的 **open** 语句
- 其次，访问结果集的语句

如果正在使用 SQL 记录来作为 I/O 对象，则记录名足以使一种类型的语句与另一种类型的语句相关，除非您将与记录相关联的 SQL 语句修改为检索不同的列集合以便在不同语句中进行更新。在这种情况下，使用结果集标识来标识与 EGL **replace** 语句相关联的结果集。

相关概念

第 209 页的『SQL 支持』

相关参考

第 576 页的『replace』

第 561 页的『open』

第 533 页的『get』

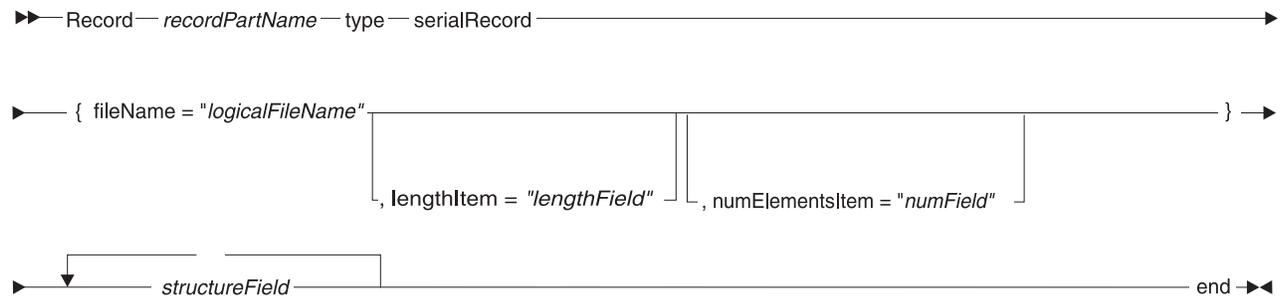
EGL 源格式的串行记录部件

可以在 EGL 文件中声明类型为 `serialRecord` 的记录部件，*EGL* 源格式对该部件作了描述。

下面是串行记录部件的示例：

```
Record mySerialRecordPart type serialRecord
{
  fileName = "myFile"
}
10 myField01 CHAR(2);
10 myField02 CHAR(78);
end
```

串行记录部件的语法图如下所示：



Record *recordPartName* **serialRecord**

将部件标识为具有 `serialRecord` 类型并指定部件名。有关命名规则，请参阅命名约定。

fileName = *"logicalFileName"*

逻辑文件名。有关输入的含义的详细信息，请参阅资源关联（概述）。有关规则，请参阅命名约定。

lengthItem = *"lengthField"*

长度字段，如支持变长记录的属性中所述。

numElementsItem = *"numField"*

元素数目字段，如支持变长记录的属性中所述。

structureField

结构字段，如 *EGL* 源格式的结构字段中所述。

相关概念

第 13 页的『EGL 项目、包和文件』

第 20 页的『对部件的引用』

第 16 页的『部件』

第 122 页的『记录部件』

第 53 页的『引用 EGL 中的变量』
第 277 页的『资源关联和文件类型』
第 25 页的『Typedef』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 68 页的『数组』
第 431 页的『EGL 源格式的 DataItem 部件』
第 448 页的『EGL 源格式』
第 481 页的『EGL 源格式的函数部件』
第 488 页的『EGL 源格式的带索引记录部件』
第 603 页的『EGL 源格式的 MQ 记录部件』
第 612 页的『命名约定』
第 31 页的『基本类型』
第 664 页的『EGL 源格式的程序部件』
第 673 页的『支持变长记录的属性』
第 676 页的『EGL 源格式的相关记录部件』
第 683 页的『EGL 源格式的 SQL 记录部件』
第 686 页的『EGL 源格式中的结构字段』

SQL 数据代码和 EGL 主变量

属性 **SQL 数据代码** 标识要与 EGL 主变量相关联的 SQL 数据类型。在声明时、验证时或生成的程序运行时，数据库管理系统使用数据代码。

您可能想改变具有基本类型 CHAR、DBCHAR、HEX 或 UNICODE 的主变量的 SQL 数据代码。然而，对于具有其它任何一种基本类型的主变量，SQL 数据代码是固定的。

如果 EGL 从数据库管理系统中检索到列定义，则不要修改检索到的 SQL 数据代码（如果有的话）。

下列各节讨论了下列主题：

- 『变长列和定长列』
- 第 681 页的『SQL 数据类型与 EGL 基本类型的兼容性』
- 第 682 页的『VARCHAR、VARGRAPHIC 和相关 LONG 数据类型』
- 第 682 页的『DATE、TIME 和 TIMESTAMP』

变长列和定长列

要指示一个表列是变长列还是定长列，可将相应主变量的 SQL 数据代码设置为适当的值，如下表所示。

EGL 基本类型	SQL 数据类型	变长或定长	SQL 数据代码
CHAR	CHAR (缺省值)	定长	453
	VARCHAR, 长度 < 255	变长	449
	VARCHAR, 长度 > 254	变长	457
DBCHAR 和 UNICODE	GRAPHIC (缺省值)	定长	469
	VARGRAPHIC, 长度 < 128	变长	465
	VARGRAPHIC, 长度 > 127	变长	473

注: SQL 数据类型可能需要使用空指示符, 但是此需求对您的 EGL 程序编码方式没有影响。有关 NULL 的详细信息, 请参阅 *SQL 支持*。

SQL 数据类型与 EGL 基本类型的兼容性

在下列任何情况下, EGL 主变量与相应的 SQL 表列是兼容的:

- SQL 列是任何形式的字符数据, 而 EGL 主变量具有 CHAR 类型并且长度小于等于 SQL 列长度。
- SQL 列是任何形式的 DBCHAR 数据, 而 EGL 主变量具有 DBCHAR 类型并且长度小于等于 SQL 列长度。
- SQL 列是任何形式的数字, 而 EGL 主变量具有下列其中一种类型:
 - BIN, 长度为 2 或 4 个字节, 并且没有小数位。
 - DECIMAL, 最大长度为 18 位, 包括小数位。对于 EGL 主变量和列, DECIMAL 变量的位数应该相同。
 - SMALLINT。
- SQL 列具有任何数据类型, EGL 主变量具有 HEX 类型, 并且列和主变量包含相同的字节数。在数据传输期间不进行数据转换。

具有 HEX 类型的 EGL 主变量支持访问具有与 EGL 基本类型不对应的数据类型的任何 SQL 列。

如果将字符数据从 SQL 表列读入到较短的主变量中, 则会截断右边的内容。要测试是否发生了截断, 请在 EGL **if** 语句中使用保留字 **trunc**。

如果将数字数据从 SQL 表列读入到较短的主变量中, 则会截断左边的前导零。如果该数字在主变量中仍放不下, 则会删除数字右边的小数部分 (十进制), 并且不发出错误指示。如果仍放不下该数字, 则会返回负数 SQL 代码, 以指示溢出状态。

下表显示了当 EGL 编辑器的检索功能从数据库管理系统中抽取信息时指定的 EGL 主变量特征。

SQL 数据类型	EGL 主变量特征			SQL 数据代码 (SQLTYPE)
	基本类型	长度	字节数	
BIGINT	HEX	16	8	493
CHAR	CHAR	1-32767	1-32767	453

SQL 数据类型	EGL 主变量特征			SQL 数据代码 (SQLTYPE)
	基本类型	长度	字节数	
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 和相关 LONG 数据类型

类型为 VARCHAR 或 VARGRAPHIC 的 SQL 表列的定义包括最大长度，检索命令使用该最大长度来指定 EGL 主变量的长度。然而，类型为 LONG VARCHAR 或 VARGRAPHIC 的 SQL 表列的定义不包括最大长度，检索命令使用 SQL 数据类型最大长度来指定长度。

DATE、TIME 和 TIMESTAMP

确保用于 EGL 系统缺省长格里历格式的格式与对 SQL 数据库管理器指定的日期格式相同。有关如何设置 EGL 格式的详细信息，请参阅 *VGVar.currentFormattedGregorianDate*。

您会想让两种格式匹配以使系统变量 *VGVar.currentFormattedGregorianDate* 提供的日期具有 SQL 数据库管理器所期望的格式。

相关概念

第 209 页的『SQL 支持』

相关参考

第 61 页的『SQL 项属性』

第 863 页的『currentFormattedGregorianDate』

SQL 记录内部结构

在下列任何情况下，都需要知道 SQL 记录的内部布局:

- 使用 EGL 赋值语句来将 SQL 记录复制至另一类型的记录，或者从另一类型的记录中复制 SQL 记录
- 传递给 EGL 程序的运行时自变量是 SQL 记录，但是程序参数不是 SQL 记录
- 传递给 EGL 函数的运行时自变量是 SQL 记录；在这种情况下，参数必须是工作存储器记录
- 作为非 EGL 程序中的参数来接收 SQL 记录

SQL 记录中的每个结构项前面都有四个字节。前两个字节是空指示符，而 null 被解释为任何负值。后面两个字节是保留用作长度字段的，不能访问该字段。

相关概念

第 130 页的『函数部件』

第 128 页的『程序部件』

第 209 页的『SQL 支持』

相关参考

第 340 页的『赋值』

EGL 源格式的 SQL 记录部件

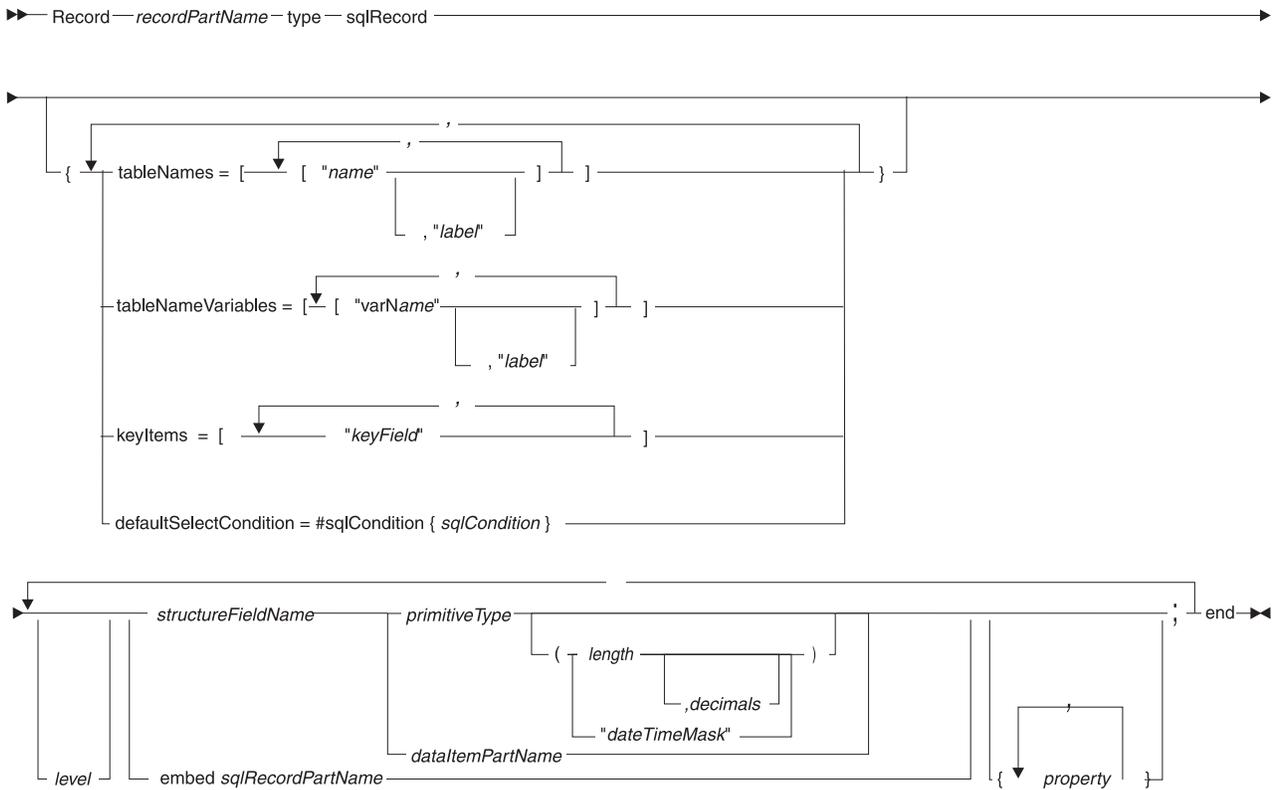
可以在 EGL 文件中声明类型为 sqlRecord 的记录部件，EGL 源格式对该部件作了描述。有关 EGL 如何与关系数据库进行交互的概述，请参阅 SQL 支持。

下面是 SQL 记录部件的示例：

```
Record mySQLRecordPart type sqlRecord
{
  tableNames = [{"mySQLTable", "T1"}],
  keyItems = ["myHostVar01"],
  defaultSelectCondition =
    #sqlCondition{ // no space between #sqlCondition and the brace
      myHostVar02 = 4 -- start each SQL comment
                    -- with a double hyphen
    }
}

// The structure of an SQL record has no hierarchy
10 myHostVar01 myDataItemPart01
{
  column = "column01",
  isNullable = no,
  isReadOnly = no
};
10 myHostVar02 myDataItemPart02
{
  column = "column02",
  isNullable = yes,
  isReadOnly = no
};
end
```

SQL 记录部件的语法图如下所示：



Record *recordPartName* sqlRecord

将该部件标识为类型为 sqlRecord 的 SQL 记录部件，并指定名称。有关规则，请参阅命名约定。

tableNames = [{"name", "label"}, ..., {"name", "label"}]

列示由 SQL 记录访问的一个或多个表。如果为给定的表名指定一个标号，则该标号包括在与该记录相关联的缺省 SQL 语句中。

通过在引号前面添加转义字符 (\)，可以将双引号 (") 包括在表名中。该约定是必需的，例如，当表名是下列其中一个 SQL 保留字时就是这种情况：

- CALL
- FROM
- GROUP
- HAVING
- INSERT
- ORDER
- SELECT
- SET
- UPDATE
- UNION
- VALUES
- WHERE

必须将那些名称中的每一个嵌入在在两对引号中。例如，如果唯一的表名是 *SELECT*，则 *tableNames* 子句如下所示：

```
tableNames=[["\"SELECT\""]]
```

当那些 SQL 保留字的其中一个被用作列名时，会发生类似的情况。

tableNameVariables = [{"varName", "label"}, ..., {"varName", "label"}]

列示一个或多个表名变量，它们中的每一个都包含由 SQL 记录访问的表的名称。仅在运行时才确定表的名称。

变量可以由库名限定，也可以带有下标。

如果为给定的表名变量指定一个标号，则该标号包括在与该记录相关联的缺省 SQL 语句中。

可以单独地使用表名变量，也可以将其与表名配合使用；但是，任何表名变量的使用都确保仅在运行时才确定 SQL 语句的特征。

通过在引号前面添加转义字符 (\)，可以将双引号 (") 包括在表名变量中。

keyItems = = [{"item", ..., "item"}]

指示与给定记录项相关联的列是数据库表中的键的一部分。如果数据库表具有组合键，则被定义为键的记录项的顺序必须与作为数据库表键的列的顺序相匹配。

defaultSelectCondition = #sqlCondition { sqlCondition }

定义隐式 SQL 语句的 WHERE 子句中的一部分搜索条件。*defaultSelectCondition* 的值不包含 SQL 关键字 WHERE。

当您编写下列其中一个 EGL 语句时，EGL 提供了带有 WHERE 子句的隐式 SQL 语句：

- **get**
- **open**
- **execute**（仅当您请求隐式 SQL DELETE 或 UPDATE 语句时）

隐式 SQL 语句并非存储在 EGL 源代码中。有关那些语句的概述，请参阅 *SQL 支持*。

level

一个整数，它指示结构字段的分层位置。如果排除此值，则该部件为记录部件；如果包括此值，则该部件为固定记录部件。

structureFieldName

结构字段的名称。有关规则，请参阅 *命名约定*。

primitiveType

指定给结构字段的基本类型。

length

结构字段的长度，它是一个整数。基于结构项的内存区的值包括指定的字符或数字的数量。

decimals

对于数字类型 (BIN、DECIMAL、NUM、NUMC 或 PACF)，可以指定 *decimals*，它是用来表示小数点后的位数的整数。最大小数位数是以下两个数字中较小的那一个：18 或声明为 *length* 的位数。小数点不与数据存储在起。

"dateTimeMask"

对于类型为 `TIMESTAMP` 和 `INTERVAL` 的项，可指定“*dateTimeMask*”，它会赋予项值中的给定位置特别的意义（如“年份位”）。掩码不会与数据存储在一起。

dataItemPartName

指定作为正在声明的结构项的格式模型的数据项部件的名称。有关详细信息，请参阅 *typeDef*。

embed *sqlRecordPartName*

指定类型为 `sqlRecord` 的记录部件的名称并将该记录部件的结构嵌入到当前记录中。嵌入的结构不会对当前记录添加层次结构级别。有关详细信息，请参阅 *typeDef*。

property

一个项属性，如 *EGL* 属性与覆盖概述中所述。在 SQL 记录中，SQL 字段属性特别重要。

相关概念

- 第 13 页的『EGL 项目、包和文件』
- 第 59 页的『EGL 属性概述』
- 第 16 页的『部件』
- 第 20 页的『对部件的引用』
- 第 122 页的『记录部件』
- 第 209 页的『SQL 支持』
- 第 25 页的『Typedef』

相关任务

- 第 690 页的『EGL 语句和命令的语法图』

相关参考

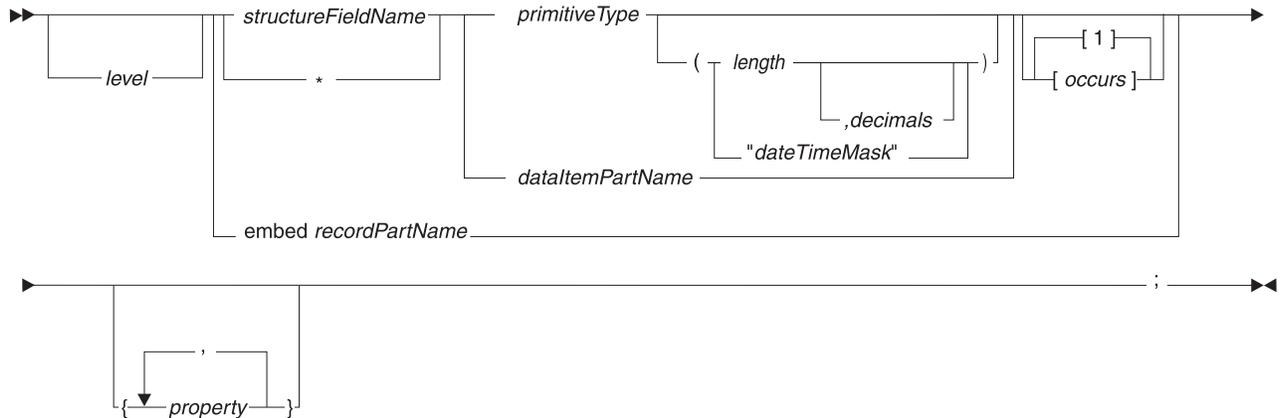
- 第 68 页的『数组』
- 第 431 页的『EGL 源格式的 DataItem 部件』
- 第 448 页的『EGL 源格式』
- 第 481 页的『EGL 源格式的函数部件』
- 第 488 页的『EGL 源格式的带索引记录部件』
- 第 603 页的『EGL 源格式的 MQ 记录部件』
- 第 612 页的『命名约定』
- 第 31 页的『基本类型』
- 第 664 页的『EGL 源格式的程序部件』
- 第 53 页的『引用 EGL 中的变量』
- 第 676 页的『EGL 源格式的相关记录部件』
- 第 679 页的『EGL 源格式的串行记录部件』
- 第 61 页的『SQL 项属性』

EGL 源格式中的结构字段

下面是结构字段的一个示例：

```
10 address;  
20 street01 CHAR(20);  
20 street02 CHAR(20);
```

结构字段的语法图如下所示:



level

一个整数，它指示结构字段的分层位置。

structureFieldName

结构字段的名称。有关规则，请参阅命名约定。

- * 指示结构字段描述一个填充符，填充符是一个名称不太重要的内存区。在对内存区的引用中，星号是无效的，如对变量和常量的引用中所述。

primitiveType

指定给结构字段的基本类型。

length

结构字段的长度，它是一个整数。基于结构字段的内存区的值包括指定的字符或数字的数量。

decimals

对于数字类型（BIN、DECIMAL、NUM、NUMC 或 PACF），可以指定 *decimals*，它是用来表示小数点后的位数的整数。最大小数位数是以下两个数字中较小的那一个：18 或声明为 *length* 的位数。小数点不与数据存储在一起来。

"dateTimeMask"

对于类型为 `TIMESTAMP` 和 `INTERVAL` 的项，可指定“*dateTimeMask*”，它会赋予字段值中的给定位置特别的意义（如“年份位”）。掩码不会与数据存储在一起来。

dataItemPartName

指定作为正在声明的结构字段的格式模型的 `dataItem` 部件的名称。有关详细信息，请参阅 *typeDef*。

embed *recordPartName*

指定记录部件的名称并将该记录部件的结构嵌入到当前记录中。嵌入的结构不会对当前记录添加层次结构级别。有关详细信息，请参阅 *typeDef*。

recordPartName

指定记录部件的名称并将该记录部件的结构包括在当前记录中。当不存在单词 *embed* 时，将记录结构作为正在声明的结构字段的子结构包括。有关详细信息，请参阅 *typeDef*。

occurs

结构项数组中的元素数。缺省值是 1，这表示除非另有指定，否则结构字段不是数组。有关详细信息，请参阅“数组”。

property

一个字段属性，如 *EGL* 属性与覆盖概述中所述。

相关概念

第 689 页的『EGL 函数的语法图』

第 59 页的『EGL 属性概述』

相关参考

第 68 页的『数组』

第 612 页的『命名约定』

第 31 页的『基本类型』

第 53 页的『引用 EGL 中的变量』

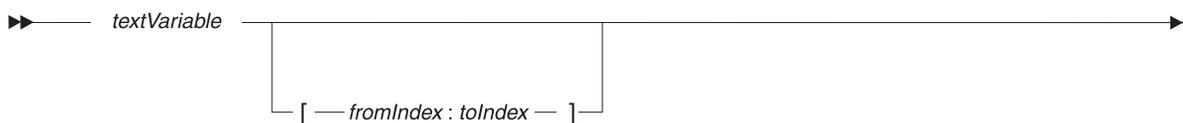
第 25 页的『Typedef』

子串

可在引用字符字段的任何上下文中引用子串，子串是该字段中的字符的顺序子集。例如，如果字段值为 *ABCD*，可以引用 *BC*，即第二个字符和第三个字符。

此外，如果目标字段的类型为 *CHAR*、*DBCHAR* 或 *UNICODE*，可在赋值语句的左边指定子串。将填充子串区域（必要时填充空格），并且被赋值的文本不会超出子串区域而是在必要时截断。

子串引用的语句如下所示。



itemReference

字符或 *HEX* 字段，但不能是文字。该项可以是系统变量或数组元素。

fromIndex

该项中您所需要的第一个字符，其中 1 表示字符项中的第一个字符，2 表示第二个字符，依此类推。可使用解析为整数的数字表达式，但该表达式不能包括函数调用。

fromIndex 的值表示字节位置，除非 *itemReference* 引用类型为 *DBCHAR* 或 *UNICODE* 的项，这种情况下该值表示双字节字符位置。

从最左边的字符算起，即使您使用双向语言（如阿拉伯语或希伯来语）也是如此。

toIndex

该项中您所需要的最后一个字符，其中 1 表示字符项中的第一个字符，2 表示第二个字符，依此类推。可使用解析为整数的数字表达式，但该表达式不能包括函数调用。

toIndex 的值表示字节位置，除非 *itemReference* 引用类型为 DBCHAR 或 UNICODE 的项，这种情况下该值表示双字节字符位置。

从最左边的字符算起，即使您使用双向语言（如阿拉伯语或希伯来语）也是如此。

相关概念

第 53 页的『引用 EGL 中的变量』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 461 页的『数字表达式』

EGL 函数的语法图

在描述给定 EGL 系统函数的主题中，语法图给出有关每个函数参数的类型及返回值的类型（如果有的话）的详细信息。函数库的名称是早前在主题中指定的。

下面是一个示例图：

```
StrLib.clip(text STRING in)  
returns (result STRING)
```

该图以函数名称开头，并且显示一系列参数说明，每个参数说明包括下列详细信息：

- 参数名称，您可以自由指定；在此示例中，该参数中名称为 *text*。
- 参数类型，这是 EGL 语言中的类型或者是若干类型的组合。（如果该类型不是 EGL 语言中的类型，将在主题中给出进一步的描述）。在此示例中，该类型为 **STRING**。
- 修饰符 **in**、**out** 或 **inOut**，如函数参数中所述。

如果参数说明是用方括号（[]）括起来的，则与该参数相关联的自变量是可选的。如果该说明是用花括号（{ }）括起来的，则自变量也是可选的，但在此情况下，可以包括多个相同类型的自变量。

如果函数返回值，则该图显示单词 *Returns* 和用圆括号括起来的名称和类型。该主题在描述返回值时引用该名称，但该名称本来是没有意义的。

如果 *returns* 子句是用方括号（[]）括起来的，则返回值是可选的。

相关参考

第 473 页的『函数调用』

第 477 页的『函数参数』

第 691 页的『EGL 库 ConsoleLib』

第 733 页的『EGL 库 J2EELib』

第 737 页的『EGL 库 JavaLib』

第 760 页的『EGL 库 LobLib』

第 767 页的『EGL 库 MathLib』

第 788 页的『EGL 库 ReportLib』

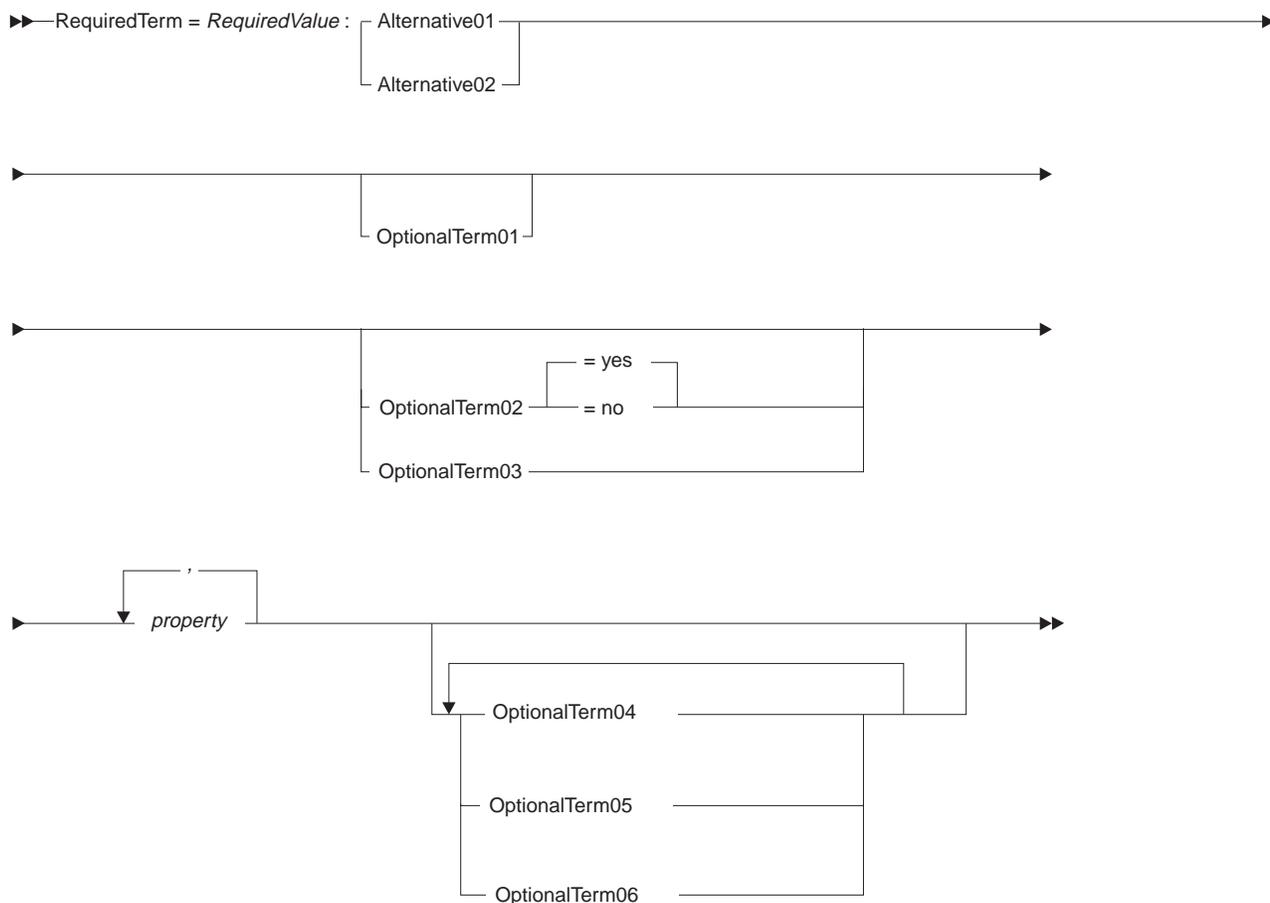
第 794 页的『EGL 库 StrLib』

第 813 页的『EGL 库 SysLib』

第 839 页的『EGL 库 VGLib』

EGL 语句和命令的语法图

IBM 语法图允许您快速了解如何构造 EGL 语句或构建命令。下面是此类语法图的一个示例:



沿着主路径，按从左到右并且从上到下的方式阅读语法图，主路径是从左边开始并带有双箭头 (>>) 的线。当沿着主路径前进时，可以选择下级路径上的条目，在这种情况下，您沿着下级路径从左到右继续进行阅读。

在示例中，主路径由四条线段组成。认识到这一点很重要。主路径的第二条和第三条线段每个都以单箭头 (>) 开始并包含下级信息。主路径的第四条线段也以单箭头 (>) 开始，包括返回箭头和下级信息，并以两个面对面的箭头 (><) 结束。

非斜体项（或符号）必须完全按照显示的方式指定。在示例中，按原样指定项 **RequiredTerm**。相反，斜体项是您指定的值的占位符。在示例中，可以包括下列任何一个符号来替换 *RequiredValue*:

```
myVariable
50
"0h!"
```

斜体项的特定需求（例如，字符串或数字是否合适）将在跟随在语法图后面的文本中进行说明，而不是在语法图本身中说明。

如果语法图显示了非字母数字字符，则输入该字符来作为语法的一部分。例如，在对 *RequiredValue* 指定值之后，输入冒号 (:) 和空格。

如果允许您从若干个项中进行选择，则这些项显示在堆栈中。在示例中，可以指定项 **Alternative01** 或 **Alternative02**。

如果必须选择堆栈中列示的那些项中的一个（就象在此例中一样），则其中一个选项（任意指定）位于堆栈的第一行上。如果不要您选择项，则所有项都位于堆栈第一行下方（**OptionalTerm01** 就是这种情况）。

位于路径上但以升高方式显示的值（= **yes** 就是这种情况）是该值所在的堆栈的缺省值。示例指示可以指定下列任何字符串，并且前两个字符串是等效的：

```
optionalTerm01 = yes
```

```
optionalTerm01
```

```
optionalTerm01 = no
```

```
OptionalTerm02
```

在项上方返回到左边的箭头指示可以重复使用该项。在示例中，您指定 *property* 的值，每个值都通过逗号与下一个值隔开。

在垂直堆栈上方返回至左边的箭头表示可以按任何顺序从条目列表中进行选择。在示例中，下列每个字符串都是有效的（其它变体也有效），但没有哪一个是必需的：

```
OptionalTerm04 OptionalTerm05
OptionalTerm06
OptionalTerm04 OptionalTerm06 OptionalTerm05
```

系统库

EGL 库 ConsoleLib

控制台库为 EGL 程序提供 consoleUI 功能。可以选择使用 **ConsoleLib** 限定符（如 **ConsoleLib.activateWindow**）。

函数	描述
activateWindow (<i>window</i>)	使指定窗口成为活动窗口，并对 ConsoleLib 变量 <i>activeWindow</i> 作相应更新。
activateWindowByName (<i>name</i>)	使指定窗口成为活动窗口，并对 ConsoleLib 变量 <i>activeWindow</i> 作相应更新。
cancelArrayDelete ()	在执行 BEFORE_DELETE OpenUI 事件代码块期间终止当前正在进行的 <i>delete</i> 操作。
cancelArrayInsert ()	在执行 BEFORE_INSERT OpenUI 事件代码块期间终止当前正在进行的 <i>insert</i> 操作。
clearActiveForm ()	清除所有字段的显示缓冲区。
clearActiveWindow ()	从活动窗口中除去所有显示内容。
clearFields ([<i>consoleField</i> {, <i>consoleField</i> })	清除活动表单中的指定字段的显示缓冲区。如果未指定任何字段，则将清除该表单的所有字段。

函数	描述
<code>clearFieldsByName (fieldName{, fieldName})</code>	清除活动表单中的指定字段的显示缓冲区。如果未指定任何字段，则将清除该表单的所有字段。
<code>clearForm (consoleForm)</code>	清除所有字段的显示缓冲区。
<code>clearWindow (window)</code>	从指定窗口中除去所有显示内容。
<code>clearWindowByName (name)</code>	从指定窗口中除去所有显示内容。
<code>closeActiveWindow ()</code>	从屏幕中清除窗口，释放与该窗口相关联的资源并激活上一个活动窗口。
<code>closeWindow (window)</code>	从屏幕中清除窗口，释放与该窗口相关联的资源并激活上一个活动窗口。
<code>closeWindowByName (name)</code>	从屏幕中清除窗口，释放与该窗口相关联的资源并激活上一个活动窗口。
<code>result = currentArrayCount ()</code>	返回与当前活动表单相关联的动态数组中的元素数目。
<code>result = currentArrayDataLine ()</code>	返回程序数组中某个程序记录的编号，在执行 OpenUI 语句期间或之后，该程序记录显示在屏幕数组的当前行中。
<code>result = currentArrayScreenLine ()</code>	返回执行 OpenUI 语句期间屏幕数组中的当前屏幕记录的编号。
<code>displayAtLine (text, line)</code>	对活动窗口中的指定空间显示字符串。
<code>displayAtPosition (text, line, column)</code>	对活动窗口中的指定空间显示字符串。
<code>displayError (msg)</code>	导致创建错误窗口并在该窗口中显示错误消息。
<code>displayFields ([consoleField{, consoleField}])</code>	对控制台显示表单字段值。
<code>displayFieldsByName (consoleFieldName{, consoleFieldName})</code>	对控制台显示表单字段值。
<code>displayForm (consoleForm)</code>	对活动窗口显示该表单。
<code>displayFormByName (formName)</code>	对活动窗口显示该表单。
<code>displayLineMode (text)</code>	以行式而不是表单/窗口方式显示字符串。
<code>displayMessage (msg)</code>	对活动窗口中的指定空间显示字符串，并使用活动窗口的 <code>messageLine</code> 设置来标识显示该字符串的位置。
<code>drawBox (row, column, depth, width)</code>	在活动窗口中按指定位置和维绘制矩形。
<code>drawBoxWithColor (row, column, depth, width, Color)</code>	在活动窗口中按指定位置、维和颜色绘制矩形。
<code>result = getKey ()</code>	从输入中读取键并返回该键的整数代码。
<code>result = getKeyCode (keyName)</code>	以字符串的形式返回指定键的键整数代码。
<code>result = getKeyName (keyCode)</code>	返回表示整数键密钥的名称。
<code>gotoField (consoleField)</code>	将光标移至指定表单字段。
<code>gotoFieldByName (name)</code>	将光标移至指定表单字段。
<code>gotoMenuItem (item)</code>	将菜单光标移至指定菜单项。
<code>gotoMenuItemByName (name)</code>	将菜单光标移至指定菜单项。
<code>hideAllMenuItems ()</code>	隐藏当前显示的菜单中的所有菜单项。

函数	描述
hideErrorWindow ()	隐藏错误窗口。
hideMenuItem (<i>item</i>)	隐藏指定菜单项以使用户不能选择该菜单项。
hideMenuItemByName (<i>name</i>)	隐藏指定菜单项以使用户不能选择该菜单项。
<i>result</i> = isCurrentField (<i>consoleField</i>)	如果光标在指定表单字段中，则返回 true ；否则返回 false 。
<i>result</i> = isCurrentFieldByName (<i>name</i>)	如果光标在指定表单字段中，则返回 true ；否则返回 false 。
<i>result</i> = isFieldModified (<i>consoleField</i>)	如果用户更改指定表单字段的内容，则返回 true ；如果返回 false ，则指示尚未编辑该字段。
<i>result</i> = isFieldModifiedByName (<i>name</i>)	如果用户更改指定表单字段的内容，则返回 true ；如果返回 false ，则指示尚未编辑该字段。
<i>result</i> = lastKeyTyped ()	返回在键盘上所按的上一个物理键的整数代码。
nextField ()	根据定义的字段遍历顺序，将光标移至下一个表单字段。
openWindow (<i>window</i>)	使窗口可视并将其添加至堆叠窗口的顶部。表单显示在窗口中。
openWindowByName (<i>name</i>)	使窗口可视并将其添加至堆叠窗口的顶部。
openWindowWithForm (<i>Window</i> , <i>form</i>)	使窗口可视并将其添加至堆叠窗口的顶部。如果在声明窗口时未定义窗口大小，则窗口大小将更改为可以容纳指定表单的大小。
openWindowWithFormByName (<i>windowName</i> , <i>formName</i>)	使窗口可视并将其添加至堆叠窗口的顶部。
previousField ()	根据定义的字段遍历顺序，将光标移至上一个表单字段。
<i>result</i> = promptLineMode (<i>prompt</i>)	在行式环境中对用户显示提示消息。
scrollDownLines (<i>numLines</i>)	将数据表滚动至数据的开头。（即，较小的记录下标）
scrollDownPage ()	将数据表滚动至数据的开头。（即，较小的记录下标）
scrollUpLines (<i>numLines</i>)	将数据表滚动至数据的结尾。（即，较大的记录下标）
scrollUpPage ()	将数据表滚动至数据的结尾（即，较大的记录下标）。
setArrayLine (<i>recordNumber</i>)	将选择移至指定程序记录。如果需要让所选记录可视，则将在屏幕中滚动数据表。
setCurrentArrayCount (<i>count</i>)	设置程序数组中存在的记录数目。必须在执行 OpenUI 语句之前调用。
showAllMenuItems ()	显示用户选择的所有菜单项。
showHelp (<i>helpkey</i>)	在执行 EGL 程序期间显示 ConsoleUI 帮助。

函数	描述
showMenuItem (<i>item</i>)	显示用户选择的指定菜单项。
showMenuItemByName(<i>name</i>)	显示用户选择的指定菜单项。

变量	描述
activeForm	活动窗口中最近显示的表单。
activeWindow	最顶部的窗口，它是未指定窗口名时执行窗口操作的目标。
commentLine	显示注释消息的窗口行。
CurrentDisplayAttrs	应用于通过显示函数显示的元素的设置。
currentRow Attrs	应用于当前行的突出显示属性。
cursorWrap	如果为 true ，则表示光标回绕至表单上的第一个字段；如果为 false ，则在光标从表单的最后一个输入字段移开时该语句结束。
defaultDisplayAttributes	新对象的表示属性的缺省设置。
defaultInputAttributes	输入操作的表示属性的缺省设置。
deferInterrupt	如果为 true ，则程序会捕获 INTR 信号并将它们记录在 <i>interruptRequested</i> 变量中，于是程序将负责监视。在 Windows 上，按下逻辑 INTERUPT 键（在缺省情况下为 CONTROL_C ）时将模拟该信号。
deferQuit	如果为 true ，则程序会捕获 QUIT 信号并将它们记录在 <i>interruptRequested</i> 变量中，于是程序将负责监视。在 Windows 上，按下逻辑 QUIT 键（在缺省情况下为 CONTROL_\\ ）时将模拟该信号。
definedFieldOrder	如果为 true ，则按向上和向下方向键将按遍历顺序移至上一个和下一个字段。如果为 false ，则按向上和向下方向键将按屏幕上的实际方向移至某个字段。
errorLine	显示错误消息的窗口。
errorWindow	在 ConsoleUI 屏幕中显示错误消息的窗口位置。
errorWindowVisible	如果为 true ，则当前将在屏幕上显示错误窗口
formLine	显示表单的窗口行。
interruptRequested	指示已经接收（或模拟） INTR 信号。
key_accept	用于成功终止 OpenUI 语句的键。缺省键为 ESCAPE 。
key_deleteLine	用于从屏幕数组中删除当前行的键。缺省键为 F2 。
key_help	用于在执行 OpenUI 语句期间显示上下文相关帮助的键。缺省键为 CTRL_W 。
key_insertLine	用于在屏幕数组中插入行的键。缺省键为 F1 。

变量	描述
key_interrupt	用于模拟 INTR 信号的键。缺省键为 CTRL_C 。
key_pageDown	用于在屏幕数组（数据表）中向前翻页的键。缺省键为 F3 。
key_pageUp	用于在屏幕数组（数据表）中向后翻页的键。缺省键为 F4 。
key_quit	用于模拟 QUIT 信号的键。缺省键为 CTRL_\
menuLine	显示菜单的窗口。
messageLine	显示消息的窗口行。
messageResource	资源束的文件名。
promptLine	显示错误消息的窗口行。
quitRequested	指示已经接收（或模拟） QUIT 信号。
screen	自动定义的缺省无边框窗口；维数等于可用显示面积的维数。
sqlInterrupt	如果为 true ，则表示用户可以中断正在处理的 SQL 语句。如果为 false ，则表示用户只能中断 OpenUI 语句。与 <i>deferInterrupt</i> 和 <i>deferQuit</i> 变量配合使用。

activeForm

系统变量 **ConsoleLib.activeForm** 是最新显示在活动窗口中的表单。

类型: ConsoleForm

相关参考

第 691 页的『EGL 库 ConsoleLib』

activateWindow()

系统函数 **ConsoleLib.activateWindow** 使指定窗口成为活动窗口并更新变量 *activeWindow*。

```
ConsoleLib.activateWindow(window1 Window inOut)
```

window1

要激活的窗口。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

activeWindow

系统变量 **ConsoleLib.activeWindow** 是显示在最上面的窗口或者最新激活的窗口。**ConsoleLib.activeWindow** 是未指定窗口名称时窗口操作的目标。

类型: 窗口

相关参考

第 691 页的『EGL 库 ConsoleLib』

activateWindowByName()

系统函数 **ConsoleLib.activateWindowByName** 使指定窗口成为活动窗口，并对 **consoleLib** 变量 *activeWindow* 作相应更新。

```
ConsoleLib.activateWindowByName(name STRING in)
```

name

窗口的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

cancelArrayDelete()

系统函数 **ConsoleLib.cancelArrayDelete** 在执行 **BEFORE_DELETE OpenUI** 事件代码块期间终止当前正在进行的 *delete* 操作。

如果在运行时，此函数将在 **OpenUI** 语句的作用域之外执行，结果成为空操作。

```
ConsoleLib.cancelArrayDelete( )
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

cancelArrayInsert()

系统函数 **ConsoleLib.cancelArrayInsert** 在执行 **BEFORE_INSERT OpenUI** 事件代码块期间终止当前正在进行的 *insert* 操作。如果在运行时，此函数将在 **OpenUI** 语句的作用域之外执行，结果成为空操作。

```
ConsoleLib.cancelArrayInsert( )
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

clearActiveForm()

系统函数 **ConsoleLib.clearActiveForm** 清除所有字段的显示缓冲区。此函数对绑定的数据元素不起作用；存储在绑定的数据元素中的数据不会被清除。

```
ConsoleLib.clearActiveForm( )
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

clearActiveWindow

系统函数 **ConsoleLib.clearActiveWindow** 从活动窗口中除去所有显示内容。这包括擦除当前表单中显示的常量信息。如果活动窗口有边框，则边框不会被擦除。该语句不会影响堆叠窗口的排序或堆叠窗口中在它之上的任何窗口。

```
ConsoleLib.clearActiveWindow( )
```

相关参考

第 691 页的『EGL 库 ConsoleLib』

clearFields()

系统函数 **ConsoleLib.clearFields** 清除指定字段的显示缓冲区。如果未指定任何字段，将清除所有字段。此函数对绑定的数据元素不起作用；存储在绑定的数据元素中的数据不会被清除。

```
ConsoleLib.clearFields(  
  [consoleField1 ConsoleField inOut  
  { , consoleField1 ConsoleField inOut}  
  ] )
```

consoleField1

类型为 ConsoleField 的变量的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

clearFieldsByName()

系统函数 **ConsoleLib.clearFieldsByName** 清除屏幕上的指定字段；如果未指定任何字段，将清除所有字段。与屏幕上的字段绑定的变量不受影响。

```
ConsoleLib.clearFieldsByName(  
  [fieldName STRING in  
  { , fieldName STRING in}] )
```

fieldName

ConsoleField 名称字段的值。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

clearForm()

系统函数 **ConsoleLib.clearForm** 清除指定表单中的所有字段。与这些字段绑定的变量不受影响。

```
ConsoleLib.clearForm(consoleForm ConsoleForm inOut)
```

consoleForm

类型为 ConsoleForm 的变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

clearWindow()

系统函数 **ConsoleLib.clearWindow** 从指定窗口中除去所有显示内容。这包括擦除当前表单中显示的常量信息。如果窗口有边框，则边框不会被擦除。该语句不会影响堆叠窗口的排序或堆叠窗口中在它之上的任何窗口。

```
ConsoleLib.clearWindow(window1 Window inOut)
```

window1

要清除的窗口。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

clearWindowByName()

系统函数 **ConsoleLib.clearWindowByName** 从指定窗口中除去所有显示内容。这包括擦除当前表单中显示的常量信息。如果窗口有边框，则边框不会被擦除。该语句不会影响堆叠窗口的排序。**ActiveWindow** 变量指的是堆叠窗口中最上面的窗口。

```
ConsoleLib.cleaWindowByName(name STRING in)
```

name

窗口的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

closeActiveWindow()

系统函数 **ConsoleLib.closeActiveWindow** 从屏幕中清除窗口，释放与该窗口相关联的资源并激活上一个活动窗口。

在调用 **ConsoleLib.closeActiveWindow** 之后，**ConsoleLib.openWindow** 或 **ConsoleLib.openWindowByName** 不能重新打开该窗口。此外，系统不允许关闭 SCREEN 窗口。

```
ConsoleLib.closeActiveWindow( )
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

closeWindow()

控制台库函数 **ConsoleLib.closeWindow** 从屏幕中清除指定窗口，释放与被清除窗口相关联的资源并激活上一个活动窗口。

在调用 **ConsoleLib.closeWindow** 之后，**ConsoleLib.openWindow** 或 **ConsoleLib.openWindowByName** 不能重新打开该窗口。此外，系统不允许关闭 SCREEN 窗口。

```
ConsoleLib.closeWindow(window1 Window inOut)
```

window1

屏幕上的指定窗口对象。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

closeWindowByName()

系统函数 **ConsoleLib.closeWindowByName** 从屏幕中清除指定窗口，释放与被关闭窗口相关联的资源并激活上一个活动窗口。

在调用 **ConsoleLib.closeWindowByName** 之后，**ConsoleLib.openWindow** 或 **ConsoleLib.openWindowByName** 不能重新打开该窗口。控制台窗口仍然处于打开状态。

```
ConsoleLib.closeWindowByName(name STRING in)
```

name

窗口的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

commentLine

系统变量 **ConsoleLib.commentLine** 是显示注释消息的窗口行。

类型: 整数

相关参考

第 691 页的『EGL 库 ConsoleLib』

currentArrayCount()

系统函数 **ConsoleLib.currentArrayCount** 返回与当前活动表单相关联的动态数组中的元素数目。

因为此函数用于帮助使用 Informix 4GL 编写的迁移应用程序, 所以建议您不要使用此函数。改为使用特定于数组的函数 **getSize**, 如数组中所述。

```
ConsoleLib.currentArrayCount( )  
returns (result INT)
```

result

元素数目。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 68 页的『数组』

第 691 页的『EGL 库 ConsoleLib』

currentArrayDataLine()

系统函数 **ConsoleLib.currentArrayDataLine** 返回程序数组中某个程序记录的编号, 在执行 **openUI** 语句期间或之后, 该程序记录显示在屏幕数组的当前行中。

```
ConsoleLib.currentArrayDataLine( )  
returns (result INT)
```

result

任何整数。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

currentArrayScreenLine()

系统函数 **ConsoleLib.currentArrayScreenLine** 返回执行 **openUI** 语句期间屏幕数组中的当前屏幕记录的编号。

```
ConsoleLib.currentArrayScreenLine( )  
returns (result INT)
```

result

任何整数。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

currentDisplayAttrs

系统变量 **ConsoleLib.currentDisplayAttrs** 指定将在设置变量之后显示的任何文本的显示特征。

类型为 `PresentationAttributes` 的变量包括的字段有 `color`、`intensity` 和 `highlight`。有关详细信息，请参阅 *ConsoleUI 部件和相关变量*。

类型: `PresentationAttributes`

相关参考

第 691 页的『EGL 库 ConsoleLib』

第 165 页的『ConsoleUI 部件和相关变量』

currentRowAttrs

系统变量 **ConsoleLib.currentRowAttrs** 是应用于屏幕数组的当前行的突出显示属性。

类型为 `PresentationAttributes` 的变量包括的字段有 `color`、`intensity` 和 `highlight`。有关详细信息，请参阅 *ConsoleUI 部件和相关变量*。

类型: `PresentationAttributes`

相关参考

`currentDisplayAttrs`

第 691 页的『EGL 库 ConsoleLib』

cursorWrap

系统变量 **ConsoleLib.cursorWrap** 指示光标是否回绕至表单上的第一个字段。如果光标回绕至表单上的第一个字段，则此函数返回 **true**；如果光标从表单上的最后一个输入字段移开时该语句结束，则返回 **false**。

类型: 布尔值

相关参考

第 691 页的『EGL 库 ConsoleLib』

defaultDisplayAttributes

系统变量 **ConsoleLib.defaultDisplayAttributes** 包含用于变量中的 *PresentationAttributes* 的设置。

类型为 `PresentationAttributes` 的变量包括的字段有 `color`、`intensity` 和 `highlight`。有关详细信息，请参阅 *ConsoleUI 部件和相关变量*。

类型: `PresentationAttributes`

相关参考

第 691 页的『EGL 库 ConsoleLib』

defaultInputAttributes

系统变量 **ConsoleLib.defaultInputAttributes** 包含输入操作的表示属性的缺省设置。

类型为 **PresentationAttributes** 的变量包括的字段有 **color**、**intensity** 和 **highlight**。有关详细信息，请参阅 *ConsoleUI 部件和相关变量*。

类型: **PresentationAttributes**

相关参考

第 691 页的『EGL 库 ConsoleLib』

deferInterrupt

控制台用户界面库变量 **ConsoleLib.deferInterrupt** 标识应用程序接收到 **INTERRUPT** 信号时的行为。如果结果为 **true**，则程序会捕获 **INTR** 信号并将它们记录在 *interruptRequested* 变量中，于是程序将负责监视。在 Windows 上，按下逻辑 **INTERRUPT** 键（在缺省情况下为 **CONTROL_C**）时将模拟该信号。如果结果为 **false**，则程序将在用户按 **interrupt** 键时结束。

类型: 布尔值

相关参考

第 691 页的『EGL 库 ConsoleLib』

deferQuit

对于系统变量 **ConsoleLib.deferQuit**，如果为 **true**，则程序会捕获 **QUIT** 信号并将它们记录在 *quitRequested* 变量中，于是程序将负责监视。在 Windows 上，按下逻辑 **QUIT** 键（在缺省情况下为 **CONTROL_**\）时将模拟该信号。如果为 **false**，则接收退出信号将终止该应用程序。

类型: 布尔值

相关参考

第 691 页的『EGL 库 ConsoleLib』

definedFieldOrder

控制台用户界面变量 **ConsoleLib.definedFieldOrder** 确定用表单输入时向上 / 向上方向键的行为。如果为 **true**，则光标会按使用向上 / 向下方向键定义的顺序来遍历字段。如果为 **false**，光标将根据屏幕上的字段的实际安排上下移动。

类型: 布尔值

相关参考

第 691 页的『EGL 库 ConsoleLib』

displayAtLine()

系统函数 **ConsoleLib.displayAtLine** 对活动窗口中的指定空间显示字符串。

```
ConsoleLib.displayAtLine(  
    text STRING in  
    line INT in)
```

text

要显示的字符串。

line

要显示字符串的行的编号。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

displayAtPosition()

系统函数 **ConsoleLib.displayAtPosition** 对活动窗口中的指定空间显示字符串。

```
ConsoleLib.displayAtPosition(  
    text STRING in,  
    line INT in,  
    column INT in)
```

text

要显示的字符串。

line

要显示字符串的行的编号。

column

要显示字符串的列的编号。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

displayError()

系统函数 **ConsoleLib.displayError** 导致创建错误窗口并在该窗口中显示错误消息。错误窗口浮动在所有其它窗口之上，直到通过调用 *hideErrorWindow()* 或按下某个键来关闭它为止。如果适用，终止铃将会被激活。

```
ConsoleLib.displayError(msg STRING in)
```

msg

要显示的错误消息。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

displayFields()

系统函数 **ConsoleLib.displayFields** 对控制台显示表单字段值。如果数据元素与字段绑定，则将按使用表单字段指定的规则从这些元素中检索数据并定义数据的格式。对于未绑定的表单字段，可通过访问 **ConsoleField.value** 字段直接对这些字段设置数据。

```
ConsoleLib.displayFields(  
  [consoleField1 ConsoleField in  
  {, consoleField1 ConsoleField in}  
  ])
```

consoleField1

类型为 ConsoleField 的变量的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

displayFieldsByName()

系统函数 **ConsoleLib.displayFieldsByName** 对控制台显示表单字段值。如果数据元素与字段绑定，则将按使用表单字段指定的规则从这些元素中检索数据并定义数据的格式。对于未绑定的表单字段，可通过访问 **ConsoleField.value** 字段直接对这些字段设置数据。

```
ConsoleLib.displayFieldsByName(  
  consoleFieldName1 ConsoleFieldName in  
  {, consoleFieldName1 ConsoleFieldName in})
```

consoleFieldName1

要显示的字段名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

displayForm()

系统函数 **ConsoleLib.displayForm** 对活动窗口显示指定表单。

```
ConsoleLib.displayForm(consoleForm ConsoleForm in)
```

consoleForm

类型为 ConsoleForm 的变量的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

displayFormByName()

系统函数 **ConsoleLib.displayFormByName** 对活动窗口显示指定表单。

```
ConsoleLib.displayFormByName(formName STRING in)
```

formName

ConsoleForm 名称字段的值。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

displayLineMode()

系统函数 **ConsoleLib.displayLineMode** 以行式而不是表单 / 窗口方式显示指定字符串。字符串值将发送至正在运行的系统上的标准输出位置。所有显示特征（如换行和滚动）由标准输出界面确定。

```
ConsoleLib.displayLine(text STRING in)
```

text

要显示的字符串。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

displayMessage()

系统函数 **ConsoleLib.displayMessage** 对活动窗口的消息行显示字符串。该函数使用活动窗口的 *MessageLine* 设置来指定显示字符串的位置。

```
ConsoleLib.displayMessage(msg STRING in)
```

msg

要显示的消息。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

drawBox()

系统函数 **ConsoleLib.drawBox** 在活动窗口中从左上角开始绘制矩形，*row*, *column* 对应头两个整数，*depth*, *width* 对应接下来的两个整数。*row* 和 *column* 与当前窗口的左上角相关联。

```
ConsoleLib.drawBox(  
    row INT in,  
    column INT in,  
    depth INT in,  
    width INT in)
```

row

相对于窗口左上角的行号。

column

相对于窗口左上角的列号。

depth

框的深度或高度。

width

框的宽度。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

drawBoxWithColor()

系统函数 **ConsoleLib.drawBoxWithColor** 在活动窗口中从左上角开始绘制矩形，*row*，*column* 对应头两个整数，*depth*，*width* 对应接下来的两个整数。*row* 和 *column* 与当前窗口的左上角相关联。将以指定颜色绘制矩形。

```
ConsoleLib.drawBoxWithColor(  
    row INT in,  
    column INT in,  
    depth INT in,  
    width INT in,  
    color enumerationColorKind in)
```

row

相对于窗口左上角的行号。

column

相对于窗口左上角的列号。

depth

框的深度或高度。

width

框的宽度。

color

框的颜色。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

errorLine

控制台用户界面变量 **ConsoleLib.errorLine** 控制在 **ConsoleUI** 屏幕中显示错误消息的行位置。

类型: INT

相关参考

第 691 页的『EGL 库 ConsoleLib』

errorWindow

系统变量 **ConsoleLib.errorWindow** 是显示来自 **ConsoleLib.displayError()** 的错误消息的窗口。

类型: 窗口

相关参考

第 691 页的『EGL 库 ConsoleLib』

第 703 页的『displayError()』

errorWindowVisible

控制台用户界面变量 **ConsoleLib.errorWindowVisible** 标识错误消息窗口的状态。如果为 **true**，窗口是可视的。如果为 **false**，则窗口不可视。

类型: 布尔值

相关参考

第 691 页的『EGL 库 ConsoleLib』

formLine

系统变量 **ConsoleLib.formLine** 是在窗口中显示表单的缺省行位置。在打开窗口时，这会影响窗口的属性。

类型: INT

相关参考

第 691 页的『EGL 库 ConsoleLib』

getKey()

系统函数 **ConsoleLib.getKey** 等待用户按下某个键，然后返回所按的物理键的整数代码。此函数从输入读取键。代码直接针对普通字符，即 Unicode 代码点。通过将结果与 **getKeyCode(String keyname)** 返回的值进行比较，可以很方便地解释结果。

```
ConsoleLib.getKey( )  
returns (result INT)
```

result

表示所按的键的整数。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

第 707 页的『getKey()』

getKeyCode()

系统函数 **ConsoleLib.getKeyCode** 返回指定键名的键整数代码。

```
ConsoleLib.getKeyCode(keyName STRING in)  
returns (result INT)
```

result

表示键名的整数。

keyName

用于计算相应键代码的逻辑或物理键的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

getKeyName()

系统函数 **ConsoleLib.getKeyName** 返回表示整数键代码的键的名称。

```
ConsoleLib.getKeyName(keyCode INT in)  
returns (result STRING)
```

result

整数键代码的键名。

keyCode

键整数代码。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

gotoField()

系统函数 **ConsoleLib.gotoField** 将光标移至指定表单字段。此函数在充当控制台表单的 **OpenUI** 语句中有效。

```
ConsoleLib.gotoField(consoleField1 ConsoleField in)
```

consoleField1

光标要移至的类型为 **ConsoleField** 的变量的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

gotoFieldByName()

系统函数 **ConsoleLib.gotoFieldByName** 将光标移至指定表单字段。此函数在充当控制台表单的 **openUI** 语句中有效。

```
ConsoleLib.gotoFieldByName(name STRING in)
```

name

光标要移至的字段名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

gotoMenuItem()

系统函数 **ConsoleLib.gotoMenuItem** 将菜单光标移至指定菜单项。调用该函数时，将选择指定的菜单项。

```
ConsoleLib.gotoMenuItem(item MenuItem in)
```

item

光标要移至的菜单项。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

gotoMenuItemByName()

系统函数 **ConsoleLib.gotoMenuItemByName** 将菜单光标移至指定菜单项。调用该函数时，将选择指定的菜单项。

```
ConsoleLib.gotoMenuItemByName(name STRING in)
```

name

光标要移至的菜单项的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

hideAllMenuItems()

系统函数 **ConsoleLib.hideAllMenuItems** 隐藏当前显示的菜单中的所有菜单项。

```
ConsoleLib.hideAllMenuItems( )
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

hideErrorWindow()

系统函数 **ConsoleLib.hideErrorWindow** 隐藏错误窗口。

```
ConsoleLib.hideErrorWindow( )
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

hideMenuItem()

系统函数 **ConsoleLib.hideMenuItem** 隐藏指定菜单项以使用户不能选择它。缺省情况下将显示所有菜单项。不能通过击键来激活隐藏的菜单项。

```
ConsoleLib.hideMenuItem(item MenuItem in)
```

item

要隐藏的菜单项。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

hideMenuItemByName()

系统函数 **ConsoleLib.hideMenuItemByName** 隐藏指定菜单项以使用户不能选择它。缺省情况下将显示所有菜单项。不能通过击键来激活隐藏的菜单项。

```
ConsoleLib.hideMenuItemByName(name STRING in)
```

name

要隐藏的菜单项的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

interruptRequested

控制台用户界面变量 **ConsoleLib.interruptRequested** 指示是否已接收或模拟 **INTR** 信号。如果为 **true**，则表示已接收 **INTR** 信号。如果为 **false**，则表示未接收 **INTR** 信号。

类型: 布尔值

相关参考

第 691 页的『EGL 库 ConsoleLib』

isCurrentField()

如果光标在字段中, 则系统函数 **ConsoleLib.isCurrentField** 返回 **yes**, 如果光标不在字段中, 则返回 **no**。此函数在充当 `arrayDictionary` 的 **OpenUI** 语句中有效。

```
ConsoleLib.isCurrentField(consoleField1 ConsoleField in)  
returns (result BOOLEAN)
```

result

如果光标在指定表单字段中, 则为 **true**; 否则为 **false**。

consoleField1

类型为 `ConsoleField` 的变量的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

isCurrentFieldByName()

如果光标在字段中, 则系统函数 **ConsoleLib.isCurrentFieldByName** 返回 **yes**, 否则返回 **no**。

此函数在充当控制台表单的 **OpenUI** 语句中有效。

```
ConsoleLib.isCurrentFieldByName(name STRING in)  
returns (result BOOLEAN)
```

result

如果光标在指定表单字段中, 则为 **true**; 否则为 **false**。

name

`ConsoleField` 名称字段的值。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

isFieldModified()

系统函数 **ConsoleLib.isFieldModified** 标识对于 **OpenUI** 表单/字段, 在当前 **OpenUI** 语句期间是否修改了字段。对于 **OpenUI** 屏幕数组 (`arrayDictionary`), 它返回的值指示自光标进入当前行后该行中的字段是否已修改。

此函数在修改字段的命令上有效, 并且不会受 **BEFORE_OPENUI** 子句中出现的语句的影响。可在这些子句中对字段赋值并让这些字段保持原样。

ConsoleLib.isFieldModified(*consoleFiled1* ConsoleField *in*)
returns (*result* BOOLEAN)

result

如果修改了指定表单字段，则为 **true**；否则为 **false**。

consoleFiled1

类型为 ConsoleField 的变量的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

isFieldModifiedByName()

系统函数 **ConsoleLib.isFieldModifiedByName** 标识指定字段的内容是否已修改。如果用户更改了字段内容，则 **ConsoleLib.isFieldModifiedByName** 返回 **yes**，如果用户未更改字段内容，则返回 **no**。

此函数在修改字段的命令上有效，并且不会受 **BEFORE_OPENUI** 子句中出现的语句的影响。可在这些子句中对字段赋值并让这些字段保持原样。

ConsoleLib.isFieldModifiedByName(*name* STRING *in*)
returns (*result* BOOLEAN)

result

如果修改了指定表单字段，则为 **true**；否则为 **false**。

name

ConsoleField 名称字段的值。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

key_accept

系统变量 **ConsoleLib.key_accept** 是用于成功终止 **OpenUI** 语句的键。缺省键为 **Esc**。

类型: CHAR(32)

相关参考

第 691 页的『EGL 库 ConsoleLib』

key_deleteLine

系统变量 **ConsoleLib.key_deleteLine** 是用于从控制台表单中的 `arrayDictionary` 删除当前行的键。缺省键为 **F2**。

类型: CHAR(32)

相关参考

第 691 页的『EGL 库 ConsoleLib』

key_help

系统变量 **ConsoleLib.key_help** 是用于在执行 **OpenUI** 语句期间显示上下文相关帮助的键。缺省键为 **CRTL_W**。

类型: CHAR(32)

相关参考

第 691 页的『EGL 库 ConsoleLib』

key_insertLine

系统变量 **ConsoleLib.key_insertLine** 标识用于在 consoleForm 上的 arrayDictionary 中插入一行的击键。缺省键为 **F1**。

类型: CHAR(32)

相关参考

第 691 页的『EGL 库 ConsoleLib』

key_interrupt

系统变量 **ConsoleLib.key_interrupt** 是用于模拟中断的键。缺省键为 **CTRL_C**。

类型: CHAR(32)

相关参考

第 691 页的『EGL 库 ConsoleLib』

key_pageDown

系统变量 **ConsoleLib.key_pageDown** 是让页面在控制台表单上的 arrayDictionary 中前进的键。缺省键为 **F3**。

类型: CHAR(32)

相关参考

第 691 页的『EGL 库 ConsoleLib』

key_pageUp

系统变量 **ConsoleLib.key_pageUp** 是让页面在控制台表单上的 arrayDictionary 中后退的键。缺省键为 **F4**。

类型: CHAR(32)

相关参考

第 691 页的『EGL 库 ConsoleLib』

key_quit

系统变量 **ConsoleLib.key_quit** 是用于在不验证用户输入的情况下离开程序的键。缺省键为 **CTRL_**。

类型: CHAR(32)

相关参考

第 691 页的『EGL 库 ConsoleLib』

lastKeyTyped()

系统函数 **ConsoleLib.lastKeyTyped** 返回在键盘上所按的上一个物理键的整数代码。

```
ConsoleLib.lastKeyTyped( )  
returns (result INT)
```

result

表示最后所按的键的整数。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

menuLine

系统变量 **ConsoleLib.menuLine** 包含在窗口中显示菜单的行位置。在打开窗口时，这会影响窗口的属性。

类型: INT

相关参考

第 691 页的『EGL 库 ConsoleLib』

messageLine

系统变量 **ConsoleLib.messageLine** 是显示消息的窗口位置。

类型: INT

相关参考

第 691 页的『EGL 库 ConsoleLib』

messageResource

系统变量 **ConsoleLib.messageResource** 是从中装入帮助和其它消息的资源束的文件名。如果此变量没有值，则 EGL 运行时会检查 Java 运行时属性 **vgj.messages.file** 中标识的文件。

类型: CHAR(255)

相关概念

第 689 页的『EGL 函数的语法图』

第 163 页的『控制台用户界面』

相关参考

第 165 页的『ConsoleUI 部件和相关变量』

第 691 页的『EGL 库 ConsoleLib』

第 493 页的『Java 运行时属性（详细信息）』

nextField()

系统函数 **ConsoleLib.nextField** 按定义的字段遍历顺序将光标移至下一个表单字段。此函数在充当控制台表单的 **openUI** 语句中有效。

```
ConsoleLib.nextField( )
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

openWindow()

系统函数 **ConsoleLib.openWindow** 使窗口可视并将其添加至堆叠窗口的顶部。

```
ConsoleLib.openWindow(window1 Window inOut)
```

window1

类型为 Window 的变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

openWindowByName()

系统函数 **ConsoleLib.openWindowByName** 使窗口可视并将其添加至堆叠窗口的顶部。

```
ConsoleLib.openWindowByName(name STRING in)
```

name

窗口名称字段的值。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

openWindowWithForm()

系统函数 **ConsoleLib.openWindowWithForm** 使窗口可视并将其添加至堆叠窗口的顶部，同时在窗口中显示表单。窗口大小将调整为能够容纳表单的大小。

```
ConsoleLib.openWindowWithForm(  
    window1 Window inOut,  
    form ConsoleForm in)
```

window1

类型为 Window 的变量。

form

类型为 ConsoleForm 的变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

openWindowWithFormByName()

系统函数 **ConsoleLib.openWindowWithFormByName** 激活窗口，使其可视并显示指定控制台表单。窗口大小将调整为能够容纳表单的大小。

```
ConsoleLib.openWindowWithFormByName(  
    windowName STRING in,  
    formName STRING in)
```

windowName

窗口名称字段的值。

formName

ConsoleForm 名称字段的值。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

previousField()

系统函数 **ConsoleLib.previousField** 按定义的字段跳进顺序将光标移至上一个表单字段。

```
ConsoleLib.previousField( )
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

promptLine

系统变量 **ConsoleLib.promptLine** 是在窗口中显示提示的缺省行。在打开窗口时，这会影响窗口的属性。

类型: INT

相关参考

第 691 页的『EGL 库 ConsoleLib』

promptLineMode()

系统函数 **ConsoleLib.promptLineMod** 以行式显示字符串并且等待用户输入，用户输入是在用户按 **Enter** 键时提交的。

```
ConsoleLib.promptLineMode(message String in)  
returns (result STRING)
```

result

用户输入。

message

要显示的短语。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

quitRequested

系统变量 **ConsoleLib.quitRequested** 指示已经接收到 **QUIT** 信号。如果为 **true**，则表示接收到 **QUIT** 信号。如果为 **false**，则表示未接收 **QUIT** 信号。

类型: 布尔值

相关参考

第 691 页的『EGL 库 ConsoleLib』

screen

系统变量 **ConsoleLib.screen** 自动定义缺省的无边框窗口。屏幕的维数等于可用显示面积的维数。

类型: 窗口

相关参考

第 691 页的『EGL 库 ConsoleLib』

scrollDownLines()

系统函数 **ConsoleLib.scrollDownLines** 将屏幕上的数据向数据底部滚动。

```
ConsoleLib.scrollDownLines(numLines INT in)
```

numLines

要向下滚动的行数。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

scrollDownPage()

系统函数 **ConsoleLib.scrollDownPage** 将屏幕上数据一页一页地向数据底部滚动。

```
ConsoleLib.scrollDownPage( )
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

scrollUpLines()

系统函数 **ConsoleLib.scrollUpLine** 将屏幕上的数据向数据顶部滚动。

```
ConsoleLib.scrollUpLines(numLines INT in)
```

numLines

要上滚的行数。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

scrollUpPage()

系统函数 **ConsoleLib.scrollUpPage** 将屏幕上的数据一页一页地向数据顶部滚动。

```
ConsoleLib.scrollUpPage( )
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

setArrayLine()

系统函数 **ConsoleLib.setArrayLine** 将选择移至指定程序记录。如果需要，将滚动数据表以便让所选记录可视。

```
ConsoleLib.setArrayLine(recordNumber INT in)
```

recordNumber

要选择的记录。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

setCurrentArrayCount()

系统函数 **ConsoleLib.setCurrentArrayCount** 指定与屏幕上的 `arrayDictionary` 绑定的动态数组中的行数。仅当您在发出使用 `arrayDictionary` 的 `openUI` 语句之前调用此函数时，此函数才有用。

```
ConsoleLib.setCurrentArrayCount(count INT in)
```

count

`openUI` 语句开始运行时数组条目的数目。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

第 565 页的『openUI』

showAllMenuItems()

系统函数 **ConsoleLib.showAllMenuItems** 显示当前显示的菜单中的所有菜单项。

```
ConsoleLib.showAllMenuItems( )
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

showHelp()

系统函数 **ConsoleLib.showHelp** 显示帮助消息。字符串自变量是资源束中使用 **ConsoleLib.messageResource** 字段配置的消息的键。

```
ConsoleLib.showHelp(helpKey STRING in)
```

helpKey

用于查找帮助消息的文本的键。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

showMenuItem()

系统函数 **ConsoleLib.showMenuItem** 显示指定菜单项以使用户可以选择它。缺省情况下将显示所有菜单项。

`ConsoleLib.showMenuItem(item MenuItem in)`

item

要显示的菜单项。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 691 页的『EGL 库 ConsoleLib』

showMenuItemByName

系统函数 **ConsoleLib.showMenuItemByName** 显示指定菜单项以使用户可以选择它。缺省情况下将显示所有菜单项。

`ConsoleLib.showMenuItemByName(name STRING in)`

name

MenuItem 名称字段的值。

相关参考

第 691 页的『EGL 库 ConsoleLib』

sqlInterrupt

对于系统变量 **ConsoleLib.sqlInterrupt**，如果为 **yes**，则表示用户可以中断正在处理的 SQL 语句。如果为 **no**，则表示用户只能中断 **OpenUI** 语句。变量 **sqlInterrupt** 与 *deferInterrupt* 和 *deferQuit* 变量配合使用。

类型：布尔值

相关参考

第 691 页的『EGL 库 ConsoleLib』

EGL 库 ConverseLib

Converse 库提供下表中显示的函数。

功能	描述
<code>clearScreen ()</code>	清除屏幕，在程序发出文本应用程序中的 <code>converse</code> 语句之前很有用
<code>displayMsgNum (msgNumber)</code>	检索程序的消息表中的值。当下一次通过 converse 、 display 、 print 或 show 语句显示表单时，将显示该消息。
<code>result = fieldInputLength (textField)</code>	返回上次显示文本表单时用户在输入字段中输入的字符数。该数目不包括前导或结尾空格或 NULL。
<code>pageEject ()</code>	使打印表单输出前进至下一页的顶部，这在程序发出 <code>print</code> 语句之前很有用

功能	描述
validationFailed (<i>msgNumber</i>)	<ul style="list-style-type: none"> 如果在文本应用程序中的字段调用函数中进行调用，则 ConverseLib.validationFailed 将导致在处理完所有验证函数之后重新显示接收到的文本表单。最后调用的 ConverseLib.validationFailed 确定所显示的消息。 如果在验证函数之外进行调用，则 ConverseLib.validationFailed 在下次通过 converse、display、print 或 show 语句显示表单时显示指定的消息。在这种情况下，行为类似于 ConverseLib.displayMsgNum 的行为。

clearScreen()

系统函数 **ConverseLib.clearScreen** 清除屏幕，在文本应用程序中，这在程序发出 **converse** 语句之前很有用。

```
ConverseLib.clearScreen( )
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 519 页的『converse』

第 720 页的『EGL 库 ConverseLib』

displayMsgNum()

系统函数 **ConverseLib.displayMsgNum** 从程序的消息表中检索值。当下次通过 **converse**、**display**、**print** 或 **show** 语句显示表单时，将显示该消息。

如果有可能的话，消息显示在表单本身中，并显示在表单属性 **msgField** 所引用的字段中。如果表单属性 **msgField** 没有值，则消息显示在表单之前，并在独立的模式屏幕上或可打印的页上显示。

ConverseLib.displayMsgNum 将一个值作为它的唯一自变量，该值将与程序消息表第一列中的每个单元格作比较，该消息表是程序的 **msgTablePrefix** 属性所引用的数据表。该函数检索到的消息位于同一行的第二列中。

```
ConverseLib.displayMsgNum(msgNumber INT in)
```

msgNumber

将按消息号从消息表中检索消息。此自变量必须是整数文字，或者是具有基本类型 **SMALLINT** 或 **INT** 或者等同的 **BIN** 类型的项。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 720 页的『EGL 库 ConverseLib』

fieldInputLength()

系统函数 **ConverseLib.fieldInputLength** 返回上次显示文本表单时用户在输入字段中输入的字符数。该数目不包括前导或结尾空格或 NULL。

如果字段处于最初定义的状态，则此函数返回长度 0。例如，如果字段包含 *value* 属性，并且在执行期间没有以任何方式对其进行修改，则计算出的长度为 0。*set form initial* 语句将字段重置为它的最初定义的状态。如果字段未处于其最初定义的状态，则根据在上一个 *converse* 语句上显示的或输入的内容来计算长度。

```
ConverseLib.fieldInputLength(textField TestFormField in)  
returns(result INT)
```

result

用户输入的字符数。

textField

文本字段的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 720 页的『EGL 库 ConverseLib』

pageEject()

系统函数 **ConverseLib.pageEject** 使打印表单输出前进至下一页的顶部，这在程序发出 *print* 语句之前很有用。

```
ConverseLib.pageEject( )
```

有关打印的其它详细信息，请参阅*打印表单*。

相关概念

第 689 页的『EGL 函数的语法图』

第 144 页的『打印表单』

相关参考

第 720 页的『EGL 库 ConverseLib』

第 576 页的『*print*』

validationFailed()

按照下列两种方式的任何一种来使用系统函数 **ConverseLib.validationFailed**:

- 如果在文本应用程序中的字段调用函数中进行调用，则 **ConverseLib.validationFailed** 将导致在处理完所有验证函数之后重新显示接收到的文本表单。最后调用的 **ConverseLib.validationFailed** 确定所显示的消息。

如果有可能的话，消息显示在表单本身中，并显示在表单属性 **msgField** 所引用的字段中。如果表单属性 **msgField** 没有值，则消息显示在表单之前，并在独立的模态屏幕上显示。

- 如果在验证函数之外进行调用，则 **ConverseLib.validationFailed** 在下次通过 **converse**、**display**、**print** 或 **show** 语句显示表单时显示指定的消息。在这种情况下，行为类似于 **ConverseLib.displayMsgNum** 的行为。

在任何情况下，赋予 **ConverseLib.validationFailed** 的值都存储在系统变量 **ConverseVar.validationMsgNum** 中。

ConverseLib.validationFailed([*msgNumber* INT *in*])

msgNumber

要显示的消息的编号。此自变量必须是整数文字，或者是具有基本类型 SMALLINT 或 INT 或者 BIN 等效类型的项。此编号将与程序消息表第一列中的每个单元格作比较，该消息表是程序的 **msgTablePrefix** 属性所引用的数据表。检索到的消息位于同一行的第二列中。

缺省情况下，消息号是 9999。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 721 页的『displayMsgNum()』

第 849 页的『validationMsgNum』

第 720 页的『EGL 库 ConverseLib』

EGL 库 DateTimeLib

如下表所示，日期和时间系统变量允许您以各种格式检索系统日期和时间。

系统变量	描述
<i>result</i> = currentDate ()	包含 8 位格里历格式 (yyyyMMdd) 的当前系统日期；您可以将此系统变量指定给类型为 DATE 的变量。
<i>result</i> = currentTime ()	包含 6 位儒略历格式 (HHmmss) 的当前系统时间；您可以将此系统变量指定给类型为 TIME 的变量。
<i>result</i> = currentTimeStamp ()	包含作为 20 位儒略历格式 (yyyyMMddHHmmssffffff) 的时间戳记的当前系统时间和日期；可将此系统变量指定给类型为 TIMESTAMP 的变量。
<i>result</i> = dateOf (<i>aTimeStamp</i>)	返回从类型为 TIMESTAMP 的变量派生的日期。
<i>result</i> = dateValue (<i>dateAsString</i>)	返回对应于输入字符串的 DATE 值。
<i>result</i> = dateValueFromGregorian (<i>gregorianIntegerDate</i>)	返回对应于格里历日期的整数表示的 DATE 值。
<i>result</i> = dateValueFromJulian (<i>julianIntegerDate</i>)	返回对应于儒略历日期的整数表示的 DATE 值。
<i>result</i> = dayOf (<i>aTimeStamp</i>)	返回一个正整数，它表示从类型为 TIMESTAMP 的变量派生的月份中的某一天。
<i>result</i> = extend (<i>extensionField</i> [, <i>mask</i>])	将时间戳记、时间或日期转换为较长或较短的时间戳记值。

系统变量	描述
<i>result</i> = intervalValue (intervalAsString)	返回反映字符串常量或文字的 INTERVAL 值。
<i>result</i> = intervalValueWithPattern (intervalAsString[, intervalMask])	返回一个 INTERVAL 值，它反映字符串常量或文字并且是基于指定的时间间隔掩码构建的。
<i>result</i> = mdy (month, day, year)	返回一个 DATE 值，它是从表示日历日期的月份、日期和年份的三个整数派生的。
<i>result</i> = monthOf (aTimeStamp)	返回一个正整数，它表示从类型为 TIMESTAMP 的变量派生的月份。
<i>result</i> = timeOf ([aTimeStamp])	返回一个字符串，它表示从 TIMESTAMP 变量或系统时钟派生的一天中的时间。
<i>result</i> = timeStampFrom (tsDate tsTime)	包含作为 20 位儒略历格式 (yyyyMMddHHmmssffffff) 的时间戳记的当前系统时间和日期；可将此系统变量指定给类型为 TIMESTAMP 的变量。
<i>result</i> = timeStampValue (timeStampAsString)	返回反映字符串常量或文字的 TIMESTAMP 值。
<i>result</i> = timeStampValueWithPattern (timeStampAsString[, timeStampMask])	返回一个 TIMESTAMP 值，它反映字符串并且是基于指定的时间间隔掩码构建的。
<i>result</i> = timeValue (timeAsString)	返回反映字符串常量或文字的 TIME 值。
<i>result</i> = weekdayOf (aTimeStamp)	返回一个正整数 (0 至 6)，它表示从类型为 TIMESTAMP 的变量派生的一周中的某天。
<i>result</i> = yearOf (aTimeStamp)	返回一个正整数，它表示从类型为 TIMESTAMP 的变量派生的年份。

要设置日期、时间或时间戳记变量，可分别指定 `VGVar.currentGregorianCalendar`、`DateTimeLib.currentTime` 和 `DateTimeLib.currentTimeStamp`。返回格式字符串的函数不能用于此用途。

相关参考

第 80 页的『EGL 语句』

currentDate()

系统函数 `DateTimeLib.currentTime` 读取系统时钟并返回表示当前日历日期的 DATE 值。该函数仅返回当前日期而不返回时间。

```
DateTimeLib.currentTime ( )
  returns (result DATE)
```

result

表示当前日历日期的 DATE 值。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 38 页的『DATE』

第 723 页的『EGL 库 DateTimeLib』

currentTime()

系统函数 **DateTimeLib.currentTime** 检索 6 位儒略历格式 (HHmmss) 的当前系统时间。程序每次引用此值时, 此值都会自动地更新。

```
DateTimeLib.currentTime( )  
returns (result TIME)
```

result

表示当前系统时间的 TIME 值。

可以按照下列方式使用 **DateTimeLib.currentTime**:

- 作为赋值语句或 **move** 语句中的源
- 作为 **return** 语句中的自变量

DateTimeLib.currentTime 的特征如下所示:

基本类型

TIME

数据长度

6

跨段保存值

否

示例:

```
myTime = DateTimeLib.currentTime;
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 723 页的『EGL 库 DateTimeLib』

currentTimeStamp()

系统函数 **DateTimeLib.currentTimeStamp** 检索 20 位格式 (yyyyMMddHHmmssffffff) 的时间戳记的当前系统时间和日期。程序每次引用此值时, 此值都会自动地更新。

```
DateTimeLib.currentTimeStamp( )  
returns (result TIMESTAMP)
```

result

表示当前系统时间和日期的 TIMESTAMPvalue 值。

可以按照下列方式使用 **DateTimeLib.currentTimeStamp**:

- 作为赋值语句或 **move** 语句中的源
- 作为 **return** 语句中的自变量

DateTimeLib.currentTimeStamp 的特征如下所示:

基本类型

TIMESTAMP

数据长度

20

跨段保存值

否

示例:

```
myTimeStamp = DateTimeLib.currentTimeStamp;
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 723 页的『EGL 库 DateTimeLib』

dateOf()

系统函数 **DateTimeLib.dateOf** 返回从类型为 `TIMESTAMP` 的变量派生的 `DATE` 值。

```
DateTimeLib.dateOf(aTimeStamp TIMESTAMP in)  
returns (result DATE)
```

result

`DATE` 值。

aTimeStamp

从中派生日期的值。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 38 页的『`DATE`』

第 723 页的『EGL 库 DateTimeLib』

dateValue()

函数 **DateTimeLib.dateValue** 返回对应于字符串的 `DATE` 值。

```
DateTimeLib.dateValue(dateAsString STRING in)  
returns (result DATE)
```

result

类型为 `DATE` 的变量。

dateAsString

包含反映掩码“`yyyyMMdd`”的字符串常量或文字。有关详细信息，请参阅 `DATE`。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 38 页的『`DATE`』

第 453 页的『日期时间表达式』

dateValueFromGregorian()

函数 **DateTimeLib.dateValueFromGregorian** 返回对应于格里历日期的整数表示的 `DATE` 值。

```
DateTimeLib.dateValueFromGregorian(  
    gregorianIntegerDate INT in)  
returns (result DATE)
```

result

类型为 DATE 的变量。

gregorianIntegerDate

以 00YYYYMMDD 或 00YYMMDD 格式表示格里历日期的 VisualAge Generator 数字值。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 38 页的『DATE』

第 723 页的『EGL 库 DateTimeLib』

dateValueFromJulian()

函数 **DateTimeLib.dateValueFromJulian** 返回对应于儒略历日期的整数表示的 DATE 值。

```
DateTimeLib.dateValueFromJulian(  
    julianIntegerDate INT in)  
returns (result DATE)
```

result

类型为 DATE 的变量。

julianIntegerDate

以格式 00YYYYYDDD 或 00YYDD 表示儒略历日期的 VisualAge Generator 数字值。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 38 页的『DATE』

第 723 页的『EGL 库 DateTimeLib』

dayOf()

系统函数 **DateTimeLib.dayOf** 返回一个正整数，它表示从类型为 **TIMESTAMP** 的变量派生的某一天（1 至 7）。

```
DateTimeLib.dayOf(aTimeStamp TIMESTAMP in)  
returns (result INT)
```

result

对应于月份中的某一天的正整数。

aTimeStamp

从中派生某一天的变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 38 页的『DATE』

第 723 页的『EGL 库 DateTimeLib』

extend()

系统函数 **DateTimeLib.extend** 将时间戳记、时间或日期转换为较长或较短的时间戳记值。示例如下所示：

- 如果输入时间戳记被定义为“ddHH”（日期和小时）并且提供时间戳记掩码“ddHHmm”（日期、小时和分钟），则 **DateTimeLib.extend** 返回与该掩码相匹配的扩展值。
- 如果输入时间戳记被定义为“yyyyMMddHHmmss”（年、月、日、小时、分钟和秒）并且提供时间戳记掩码“yyyy”（年份），则 **DateTimeLib.extend** 返回与该掩码相匹配的缩短了的值。

```
DateTimeLib.extend(  
    extensionField dateOrTimeOrTimeStamp in  
    [, mask outputTimeStampMask in  
    ]  
)  
returns (result TIMESTAMP)
```

result

类型为 **TIMESTAMP** 的变量。

extensionField

类型为 **TIMESTAMP**、**TIME** 或 **DATE** 的字段名称。该字段包含要延长或缩短的值。

mask

一个字符串文字或常量，用于定义该函数返回的时间戳记值的掩码。有关详细信息，请参阅 **TIMESTAMP**。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 453 页的『日期时间表达式』

intervalValue()

日期时间值函数 **DateTimeLib.intervalValue** 返回一个 **INTERVAL** 值，该值反映字符串常量或文字，并且是根据缺省时间间隔掩码（即 **yyyyMM**）构建的。

输入字符串必须包含 6 位数字。头 4 位表示时间间隔中的年数，最后 2 位表示月数。

如果希望指定 **yyyyMM** 之外的掩码，则调用 **DateTimeLib.intervalValueWithPattern**。

```
DateTimeLib.intervalValue(intervalAsString STRING in)  
returns (result INTERVAL)
```

result

类型为 **INTERVAL** 的变量

intervalAsString

包含含义由时间间隔掩码 **yyyyMM** 指示的 6 位数字的字符串常量或文字

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 453 页的『日期时间表达式』

第 38 页的『INTERVAL』

『intervalValueWithPattern()』

intervalValueWithPattern()

日期时间值函数 **DateTimeLib.intervalValueWithPattern** 返回一个 INTERVAL 值，该值反映字符串常量或文字并可以选择基于指定的时间间隔掩码来构建。例如，如果掩码为 yyyy，则输入字符串必须包含四位数字，并且这些数字体现时间间隔中表示的年数。

```
DateTimeLib.intervalValueWithPattern(  
    intervalAsString STRING in  
    [, intervalMask STRING in  
    ])  
returns (result INTERVAL)
```

result

类型为 INTERVAL 的变量。

intervalAsString

一个字符串常量或文字，包含含义由时间间隔掩码指示的数字。

intervalMask

指定时间间隔掩码，它对第一个参数中的每个数字指定含义。缺省掩码为 yyyyMM。
有关其它详细信息，请参阅 *INTERVAL*。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 453 页的『日期时间表达式』

第 38 页的『INTERVAL』

mdy()

mdy 运算符返回一个 DATE 值，它是从表示日历日期的月份、日期和年份的三个整数派生的。

```
DateTimeLib.mdy(  
    month INT in,  
    day INT in,  
    year INT in)  
returns (result DATE)
```

result

DATE 值。

month

范围从 1 到 12 的整数，表示月份。

day

表示范围从 1 到 28、29、30 或 31（依据月份）的日期的整数。

year

表示年份的四位整数。

如果指定日历上的月份和日期范围之外的值，或者如果不是 3 个操作数，则会发生错误。必须用圆括号将三个整型表达式操作数括起来，操作数之间用逗号隔开，就好像 MDY() 是函数一样来处理。第三个表达式不能是年份的缩写。例如，99 指的是第一个世纪的年份，即大约 1900 年以前。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 38 页的『DATE』

第 723 页的『EGL 库 DateTimeLib』

monthOf()

系统函数 **DateTimeLib.monthOf** 返回一个正整数，它表示从类型为 **TIMESTAMP** 的变量派生的月份。

```
DateTimeLib.monthOf(aTimeStamp TIMESTAMP in)  
returns (result INT)
```

result

表示月份的正整数。

aTimeStamp

从中派生月份的 **TIMESTAMP** 变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 38 页的『DATE』

第 723 页的『EGL 库 DateTimeLib』

timeOf()

系统函数 **DateTimeLib.timeOf** 返回一个字符串，它表示从 **TIMESTAMP** 变量或系统时钟派生的一天中的时间。

```
DateTimeLib.timeOf([aTimeStamp TIMESTAMP in])  
returns (result STRING)
```

result

aTimeStamp 自变量的时间部分，基于 24 小时时钟和以下格式：

hh:mm:ss

hh 小时，用两位字符串表示。

mm

分钟，用两位字符串表示。

ss 秒，用两位字符串表示。

aTimeStamp

DATETIME 值。如果未指定值，则运算符返回表示当前时间的字符串。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 38 页的『DATE』

第 723 页的『EGL 库 DateTimeLib』

timeStampFrom()

函数 **DateTimeLib.timeStampFrom** 返回一个 **TIMESTAMP** 值，它是根据您指定的 **DATE** 和 **TIME** 构建的。

```
DateTimeLib.timeStampFrom(  
    tsDate DATE in,  
    tsTime TIME in)  
returns (result TIMESTAMP)
```

result

类型为 **TIMESTAMP** 的值。

tsDate

类型为 **DATE** 的变量。

tsTime

类型为 **TIME** 的变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 453 页的『日期时间表达式』

第 723 页的『EGL 库 DateTimeLib』

第 40 页的『TIMESTAMP』

timeStampValue()

函数 **DateTimeLib.timeStampValue** 返回一个 **TIMESTAMP** 值，该值反映字符串常量或文字，并且是根据缺省时间戳记掩码（即 `yyyyMMddHHmmss`）构建的。

输入字符串必须包含 14 位数字：

- 头 4 位数字表示年份
- 接下来的 2 位数字表示数字月份
- 再接下来的 2 位数字表示当月某天
- 再接下来的 2 位数字表示小时数（00 到 24）
- 再接下来的 2 位数字表示一小时内的分钟数
- 最后 2 位数字表示一分钟内的秒数

如果希望指定 `yyyyMMddHHmmss` 之外的掩码，则调用 **DateTimeLib.timestampValueWithPattern**。

```
DateTimeLib.timeStampValue(timeStampAsString STRING in)  
returns (result TIMESTAMP)
```

result

类型为 **TIMESTAMP** 的变量。

timeStampAsString

包含含义由时间戳记掩码 *yyyyMMddHHmmss* 指示的 14 位数字的字符串常量或文字

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 453 页的『日期时间表达式』

第 40 页的『TIMESTAMP』

『timeStampValueWithPattern()』

timeStampValueWithPattern()

函数 **DateTimeLib.timeStampValueWithPattern** 返回一个 **TIMESTAMP** 值，该值反映了字符串常量或文字并可以选择基于指定的时间戳记掩码来构建。例如，如果掩码为“yyyy”，则输入字符串必须包含四位数字，并且这些数字在时间戳记中表示年份值。

```
DateTimeLib.timeStampValueWithPattern(  
    timeStampAsString STRING in  
    [, timeStampMask STRING in  
    ]  
)  
returns (result TIMESTAMP)
```

result

类型为 **TIMESTAMP** 的变量。

timeStampAsString

包含含义由时间戳记掩码指示的数字的字符串常量或文字。

timeStampMask

指定时间戳记掩码，它对第一个参数中的每个数字指定含义。缺省掩码为 *yyyyMMddHHmmss*。有关其它详细信息，请参阅 **TIMESTAMP**。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 453 页的『日期时间表达式』

第 40 页的『TIMESTAMP』

timeValue()

日期时间值函数 **DateTimeLib.timeValue** 返回反映字符串常量或文字的 **TIME** 值。

```
DateTimeLib.timeValue(timeAsString STRING in)  
returns (result TIME)
```

result

类型为 **TIME** 的变量。

timeAsString

包含反映掩码“HHmmss”的数字的字符串常量或文字。有关详细信息，请参阅 **TIME**。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 453 页的『日期时间表达式』

第 40 页的『TIME』

weekdayOf()

系统函数 **DateTimeLib.weekdayOf** 返回一个正整数，它表示从类型为 **TIMESTAMP** 的变量派生的一个星期中的某一天。数字 0 表示星期日，1 表示星期一，以此类推。

```
DateTimeLib.weekdayOf(aTimeStamp TIMESTAMP in)  
returns (result INT)
```

result

0 到 6 的正整数。

aTimeStamp

从中派生一个星期中的某一天的 **TIMESTAMP** 变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 38 页的『DATE』

第 723 页的『EGL 库 DateTimeLib』

yearOf()

系统函数 **DateTimeLib.yearOf** 返回一个四位整数，它表示从类型为 **TIMESTAMP** 的变量派生的年份。

```
DateTimeLib.yearOf(aTimeStamp TIMESTAMP in)  
returns (result INT)
```

result

表示年份的整数。

aTimeStamp

从中派生年份的 **TIMESTAMP** 变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 38 页的『DATE』

第 723 页的『EGL 库 DateTimeLib』

EGL 库 J2EELib

下表列出库 J2EELib 中的系统函数。

功能	描述
<code>clearRequestAttr (key)</code>	除去请求对象中与指定键相关联的自变量。
<code>clearSessionAttr (key)</code>	除去会话对象中与指定键相关联的自变量。
<code>getRequestAttr (key, argument)</code>	使用指定的键以将请求对象中的自变量检索到指定的变量中。

功能	描述
<code>getSessionAttr (key, argument)</code>	使用指定的键以将会话对象中的自变量检索到指定的变量中。
<code>setRequestAttr (key, argument)</code>	使用指定的键以将指定的自变量放到请求对象中。
<code>setSessionAttr (key, argument)</code>	使用指定的键以将指定的自变量放到会话对象中。

clearRequestAttr()

系统函数 **J2EELib.clearRequestAttr** 除去请求对象中与指定的键相关联的自变量。在 `PageHandler` 中以及在运行于 Web 应用程序中的程序中，此函数很有用。

可以通过使用系统函数 `J2EELib.setRequestAttr` 来在请求对象中设置自变量。可以使用系统函数 `J2EELib.getRequestAttr` 来检索自变量。

```
J2EELib.clearRequestAttr(key STRING in)
```

key

类型为字符串的字符串文字或表达式

相关概念

第 689 页的『EGL 函数的语法图』

第 177 页的『PageHandler』

相关参考

第 733 页的『EGL 库 J2EELib』

第 735 页的『getRequestAttr()』

第 736 页的『setRequestAttr()』

clearSessionAttr()

系统函数 **J2EELib.clearSessionAttr** 除去会话对象中与指定的键相关联的自变量。在 `PageHandler` 中以及在运行于 Web 应用程序中的程序中，此函数很有用。

可以通过使用系统函数 `J2EELib.setSessionAttr` 来在会话对象中设置自变量。可以使用系统函数 `J2EELib.getSessionAttr` 来检索自变量。

```
J2EELib.clearSessionAttr(key STRING in)
```

key

类型为字符串的字符串文字或表达式

相关概念

第 689 页的『EGL 函数的语法图』

第 177 页的『PageHandler』

相关参考

第 733 页的『EGL 库 J2EELib』

第 735 页的『getSessionAttr()』

第 736 页的『setSessionAttr()』

getRequestAttr()

系统函数 **J2EELib.getRequestAttr** 使用指定的键来将请求对象的自变量检索到指定的变量中。在 PageHandler 中以及在运行于 Web 应用程序中的程序中，此函数很有用。

如果使用指定的键找不到对象，则目标变量不更改。如果检索到的对象具有错误的类型，则将抛出异常，并且程序或 PageHandler 终止。

可以通过使用系统函数 **J2EELib.setRequestAttr** 来在请求对象中放置自变量。自变量对象被放在 servlet 的请求集合中，只要 servlet 请求有效，自变量对象就可供访问。从页中提交表单将导致创建新请求。

```
J2EELib.getRequestAttr(  
    key STRING in,  
    argument attribute inOut)
```

key

字符文字，或者是具有任何字符类型的项。

argument

项、记录或数组。

相关概念

第 689 页的『EGL 函数的语法图』

第 177 页的『PageHandler』

相关参考

第 733 页的『EGL 库 J2EELib』

第 736 页的『setRequestAttr()』

getSessionAttr()

系统函数 **J2EELib.getSessionAttr** 使用指定的键来将会话对象的自变量检索到指定的变量中。在 PageHandler 中以及在运行于 Web 应用程序中的程序中，此函数很有用。

如果使用指定的键找不到对象，则目标变量不更改。如果检索到的对象具有错误的类型，则将抛出异常，并且程序或 PageHandler 终止。

可以通过使用系统函数 **J2EELib.setSessionAttr** 来在会话对象中放置自变量。

```
J2EELib.getSessionAttr(  
    key STRING in,  
    argument attribute in)
```

key

字符文字，或者是具有任何字符类型的项。

argument

项、记录或数组。

相关概念

第 689 页的『EGL 函数的语法图』

第 177 页的『PageHandler』

相关参考

第 733 页的『EGL 库 J2EELib』

第 736 页的『setSessionAttr()』

setRequestAttr()

系统函数 **J2EELib.setRequestAttr** 使用指定的键来将指定的自变量放到请求对象中。在 PageHandler 中以及在运行于 Web 应用程序中的程序中，此函数很有用。以后，可以使用系统函数 **J2EELib.getRequestAttr** 来检索自变量。

```
J2EELib.setRequestAttr(  
    key STRING in,  
    argument attribute in)
```

key

字符文字，或者是具有任何字符类型的项。

argument

项、记录或数组。

在生成的 Java 输出中，项自变量是作为 Java 基本对象（字符串、整数和十进制数等等）传递的。记录自变量是作为记录 bean 传递的。数组是作为具有相关类型的数组列表传递的。自变量对象被放在 servlet 的请求集合中，只要 servlet 请求有效，自变量对象就可供访问。从页中提交表单将导致创建新请求。

相关概念

第 689 页的『EGL 函数的语法图』

第 177 页的『PageHandler』

相关参考

第 733 页的『EGL 库 J2EELib』

第 735 页的『getRequestAttr()』

setSessionAttr()

系统函数 **J2EELib.setSessionAttr** 使用指定的键来将指定的自变量放到会话对象中。在 PageHandler 中以及在运行于 Web 应用程序中的程序中，此函数很有用。以后，可以使用系统函数 **J2EELib.getSessionAttr** 来检索自变量。

```
J2EELib.setSessionAttr(  
    key STRING in,  
    argument attribute in)
```

key

字符文字，或者是具有任何字符类型的项。

argument

项、记录或数组。

在生成的 Java 输出中，项自变量是作为 Java 基本对象（字符串、整数和十进制数等等）传递的。记录自变量是作为记录 bean 传递的。数组是作为具有相关类型的数组列表传递的。

相关概念

第 689 页的『EGL 函数的语法图』

第 177 页的『PageHandler』

相关参考

第 733 页的『EGL 库 J2EELib』

第 735 页的『getSessionAttr()』

EGL 库 JavaLib

下表中列示了 Java 访问函数。

功能	描述
<code>result = getField (identifierOrClass, field)</code>	返回指定的对象或类的指定字段的值
<code>result = invoke (identifierOrClass, method[, argument])</code>	调用 Java 对象或类的方法，并可以返回一个值
<code>result = isNull (identifier)</code>	返回一个值（1 表示 true，0 表示 false）以指示指定的标识是否引用空对象
<code>result = isObjID (identifier)</code>	返回一个值（1 表示 true，0 表示 false）以指示指定的标识是否在对象空间中
<code>result = qualifiedTypeName(identifier)</code>	返回对象的类在对象空间中的标准名称
<code>remove (identifier)</code>	从对象空间中除去指定的标识，并且，如果没有其它标识引用该对象，则除去该对象
<code>removeAll ()</code>	从对象空间中除去所有标识和对象
<code>setField (identifierOrClass, field, value)</code>	设置 Java 对象或类中的字段的值
<code>store (storeId, identifierOrClass, method[,argument])</code>	调用方法并将返回的对象（或 NULL）与指定的标识一起放到对象空间中
<code>storeCopy (sourceId, targetID)</code>	根据对象空间中的另一个标识来创建新标识，以使两者引用同一个对象
<code>storeField (storeId, identifierOrClass, field)</code>	将类字段或对象字段的值放到对象空间中
<code>storeNew(storeId, class[,argument])</code>	调用类的构造函数并将新对象放到对象空间中

Java 访问函数

Java 访问函数是 EGL 系统函数，它们允许生成的 Java 代码访问本机 Java 对象和类；具体地说，允许 Java 代码访问本机代码的公用方法、构造函数和字段。

在运行时，由于存在 EGL Java 对象空间而使此 EGL 功能成为可能，其中，EGL Java 对象空间是一组名称以及那些名称所引用的对象。有一个对象空间可供生成的程序以及该程序以本地方式调用的所有 Java 生成代码使用，无论那些调用是直接的还是通过另一个本地 Java 生成程序进行的均如此，并且可用于任何级别的调用。此对象空间不能用在任何本机 Java 代码中。

要在对象空间中存储和检索对象，请调用 Java 访问函数。这些调用包括使用标识，每个标识都是用来存储对象或用来匹配对象空间中已存在的名称的字符串。当标识与名称相匹配时，代码就可以访问与该名称相关联的对象。

接下来的几个部分如下所示：

- 『EGL 与 Java 类型的映射』
- 第 739 页的『示例』
- 第 742 页的『错误处理』

EGL 与 Java 类型的映射： 将把传递给方法的每个自变量（以及赋予字段的每个值）映射至 Java 对象或基本类型。例如，具有 EGL 基本类型 CHAR 的项是作为 Java 字符串类的对象传递的。提供了强制类型转换运算符来用于 EGL 类型到 Java 类型的映射并不充分的情况。

当您指定 Java 名称时，EGL 将从值的开头和末尾除去单字节和双字节空格，其中，值是区分大小写的。截断操作在任何强制类型转换之前发生。此规则适用于字符串文字以及类型为 CHAR、DBCHAR、MBCHAR 或 UNICODE 的项。除非将值强制类型转换为 objID 或 NULL，否则，在指定方法自变量或字段值时不会发生这种截断（例如，将字符串“ my data ”按原样传递给方法）。

下表描述了所有有效映射。

自变量的类别		示例	Java 类型
字符串文字或类型为 CHAR、DBCHAR、MBCHAR 或 UNICODE 的项	不进行强制类型转换	"myString"	java.lang.String
	使用 objId 进行强制类型转换，objId 指示了标识	(objId)"myId" x = "myId"; (objId)x	标识所引用的对象的类
	使用 NULL 进行强制类型转换，这可能适合于提供对标准类的空引用	(null)"java.lang.Thread" x = "java.util.HashMap"; (null)x	指定的类 注：不能传入强制类型转换为 NULL 的数组，如 (null)"int[]"
	使用 char 进行强制类型转换，这表示传递值的第一个字符（下一列中的每个示例都传递“a”）	(char)"abc" x = "abc"; (char)x	char
类型为 FLOAT 的项或浮点文字	不进行强制类型转换	myFloatValue	double
类型为 HEX 的项	不进行强制类型转换	myHexValue	byte array
类型为 SMALLFLOAT 的项	不进行强制类型转换	mySmallFloat	float
类型为 DATE 的项	不进行强制类型转换	myDate	java.sql.Date
类型为 TIME 的项	不进行强制类型转换	myTime	java.sql.Time
类型为 TIMESTAMP 的项	不进行强制类型转换	myTimeStamp	java.sql.Timestamp
类型为 INTERVAL 的项	不进行强制类型转换	myInterval	java.lang.String
浮点文字	不进行强制类型转换	-6.5231E96	double
不包含小数的数字项（或非浮点文字）；前导零计入文字的位数。	不进行强制类型转换，1-4 位	0100	short
	不进行强制类型转换，5-9 位	00100	int
	不进行强制类型转换，9-18 位	1234567890	long
	不进行强制类型转换，高于 18 位	1234567890123456789	java.math.BigInteger

自变量的类别		示例	Java 类型
包含小数的数字项 (或非浮点文字); 前导零和结尾零计入 文字的位数。	不进行强制类型转 换, 1-6 位	3.14159	float
	不进行强制类型转 换, 7-18 位	3.14159265	double
	不进行强制类型转 换, 高于 18 位	56789543.222	java.math.BigDecimal
数字项或非浮点文 字, 带有或不带小数	使用 bigdecimal、 biginteger、byte、 double、float、 short、int 或 long 进行强制 类型转换	X = 42; (byte)X (long)X	指定的基本类型; 但如 果值超出该类型的范 围, 则会丢失精度, 符 号也可能会更改
	使用布尔值进行强 制类型转换, 这表 示非零为 true, 零 为 false	X = 1; (boolean)X	boolean

注: 为避免精度降低, 使用 EGL 浮点项来表示 Java 双精度, 并使用 EGL smallfloat 项来表示 Java 浮点。使用其它 EGL 类型中的某个类型可能导致值被四舍五入。

有关 EGL 中的项的内部格式的详细信息, 请参阅有关基本类型的帮助页面。

示例: 本节给出有关如何使用 Java 访问函数的示例。

打印日期字符串: 以下示例打印日期字符串:

```
// call the constructor of the Java Date class and
// assign the new object to the identifier "date".
JavaLib.storeNew( (objId)"date", "java.util.Date" );

// call the toString method of the new Date object
// and assign the output (today's date) to the charItem.
// In the absence of the cast (objId), "date"
// refers to a class rather than an object.
charItem = JavaLib.invoke( (objId)"date", "toString" );

// assign the standard output stream of the
// Java System class to the identifier "systemOut".
JavaLib.storeField( (objId)"systemOut",
    "java.lang.System", "out" );

// call the println method of the output
// stream and print today's date.
JavaLib.invoke( (objID)"systemOut", "println", charItem );

// The use of "java.lang.System.out" as the first
// argument in the previous line would not have been
// valid, as the argument must either be a
// an identifier already in the object space or a class
// name. The argument cannot refer to a static field.
```

测试系统属性: 以下示例检索系统属性并测试值是否存在:

```
// assign the name of an identifier to an item of type CHAR
valueID = "osNameProperty"

// place the value of property os.name into the
// object space, and relate that value (a Java String)
```

```

// to the identifier osNameProperty
JavaLib.store((objId)valueId, "java.lang.System",
    "getProperty", "os.name");

// test whether the property value is non-existent
// and process accordingly
myNullFlag = JavaLib.isNull( (objId)valueId );

if( myNullFlag == 1 )
    error = 27;
end

```

使用数组: 当在 EGL 中使用 Java 数组时, 请使用 Java 类 `java.lang.reflect.Array`, 如后面的示例所示以及如 Java API 文档所述。由于 Java 数组没有构造函数, 所以不能使用 **JavaLib.storeNew** 来创建 Java 数组。

使用 `java.lang.reflect.Array` 的静态方法 `newInstance` 以在对象空间中创建数组。在创建数组之后, 使用该类中的其它方法来访问元素。

方法 `newInstance` 期望两个自变量:

- 一个类对象, 它确定正在创建的数组的类型
- 一个数字, 它指定数组中有多少个元素

根据您要创建的是对象数组还是基本类型数组, 用于标识类对象的代码会有所不同。与数组进行交互的后续代码也会以此为基础而有所变化。

使用对象数组: 以下示例显示如何创建可使用标识“myArray”来访问的 5 个元素的对象数组:

```

// Get a reference to the class, for use with newInstance
JavaLib.store( (objId)"objectClass", "java.lang.Class",
    "forName", "java.lang.Object" );

// Create the array in the object space
JavaLib.store( (objId)"myArray", "java.lang.reflect.Array",
    "newInstance", (objId)"objectClass", 5 );

```

如果要创建存放另一种类型的对象的数组, 请更改传递给 **JavaLib.store** 的第一次调用的类名。例如, 要创建字符串对象的数组, 传递“`java.lang.String`”而不是“`java.lang.Object`”。

要访问对象数组的元素, 请使用 `java.lang.reflect.Array` 的 `get` 和 `set` 方法。在以下示例中, `i` 和 `length` 是数字项:

```

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 );

// Here, process the element as appropriate
JavaLib.invoke( "java.lang.reflect.Array", "set",
    (objId)"myArray", i, (objId)"element" );
i = i + 1;
end

```

上一个示例等同于下列 Java 代码:

```

int length = myArray.length;

for ( int i = 0; i < length; i++ )
{
    Object element = myArray[i];

    // Here, process the element as appropriate

    myArray[i] = element;
}

```

使用 *Java* 基本类型的数组: 要创建用于存储 *Java* 基本类型而不是对象的数组, 请在使用 `java.lang.reflect.Array` 之前的步骤中使用另一种机制。特别是, 通过访问基本类型的静态字段 `TYPE` 来获取 `newInstance` 的 `Class` 自变量。

以下示例创建 `myArray2`, 这是包含 30 个元素的整数数组:

```

// Get a reference to the class, for use with newInstance
JavaLib.storeField( (objId)"intClass",
    "java.lang.Integer", "TYPE");

// Create the array in the object space
JavaLib.store( (objId)"myArray2", "java.lang.reflect.Array",
    "newInstance", (objId)"intClass", 30 );

```

如果要创建存放另一种类型的原语的数组, 请更改传递给 `JavaLib.storeField` 的调用的类名。例如, 要创建字符数组, 请传递“`java.lang.Character`”而不是“`java.lang.Integer`”。

要访问基本类型数组的元素, 请使用特定于基本类型的 `java.lang.reflect.Array` 方法。这样的方法包括 `getInt`、`setInt`、`getFloat` 和 `setFloat` 等等。在以下示例中, `length`、`element` 和 `i` 是数字项:

```

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 );

    // Here, process an element as appropriate

    JavaLib.invoke( "java.lang.reflect.Array", "setDouble",
        (objId)"myArray2", i, element );
    i = i + 1;
end

```

上一个示例等同于下列 *Java* 代码:

```

int length = myArray2.length;

for ( int i = 0; i < length; i++ )
{
    double element = myArray2[i];

    // Here, process an element as appropriate

    myArray2[i] = element;
}

```

使用集合: 要对名为 `list` 的变量所引用的集合进行迭代, *Java* 程序执行以下操作:

```

Iterator contents = list.iterator();

while( contents.hasNext() )
{
    Object myObject = contents.next();
    // Process myObject
}

```

假定 `hasNext` 是数字数据，并且程序将一个集合与名为 `list` 的标识相关。以下 EGL 代码等同于前面描述的 Java 代码：

```

JavaLib.store( (objId)"contents", (objId)"list", "iterator" );
hasNext = JavaLib.invoke( (objId)"contents", "hasNext" );

while ( hasNext == 1 )
    JavaLib.store( (objId)"myObject", (objId)"contents", "next");

// Process myObject
hasNext = JavaLib.invoke( (objId)"contents", "hasNext" );
end

```

将数组转换为集合： 要根据对象数组来创建集合，请使用 `java.util.Arrays` 的 `asList` 方法，如以下示例所示：

```

// Create a collection from array myArray
// and relate that collection to the identifier "list"
JavaLib.store( (objId)"list", "java.util.Arrays",
    "asList", (objId)"myArray" );

```

接着，对列表进行迭代，如前一节所示。

数组到集合的转换仅对对象数组起作用，而对 Java 基本类型数组不起作用。请注意不要将 `java.util.Arrays` 与 `java.lang.reflect.Array` 混淆。

错误处理： 如特定于函数的帮助页面所述，许多 Java 访问函数与错误代码相关联。如果发生所列示的某个错误时系统变量 `VGVar.handleSysLibraryErrors` 的值是 1，EGL 就将系统变量 `sysVar.errorCode` 设置为非零值。如果发生某个错误时 `VGVar.handleSysLibraryErrors` 的值为 0，则程序结束。

特别要注意的是 `sysVar.errorCode` 值“00001000”，这个值指示被调用方法抛出了异常，或者作为类初始化的结果而抛出了异常。

当抛出异常时，EGL 将其存储在对象空间中。如果发生另一个异常，则第二个异常将取代第一个异常。可使用标识 `caughtException` 来访问所发生的最后一个异常。

在并不常见的情况下，被调用方法抛出的不是异常而是错误，如 `OutOfMemoryError` 或 `StackOverflowError`。在这样的情况下，无论系统变量 `VGVar.handleSysLibraryErrors` 具有何值，程序都结束。

下列 Java 代码显示 Java 程序可以如何使用多个 `catch` 块来处理不同类型的异常。此代码尝试创建 `FileOutputStream` 对象。发生故障，导致代码设置 `errorType` 变量并存储所抛出的异常。

```

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;
}

```

以下 EGL 代码等同于前面的 Java 代码:

```

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

```

相关参考

- 第 737 页的『EGL 库 JavaLib』
- 第 86 页的『异常处理』
- 第 31 页的『基本类型』
- 『getField()』
- 第 748 页的『isNull()』
- 第 749 页的『isObjID()』
- 第 750 页的『qualifiedTypeName()』 第 751 页的『remove()』
- 第 752 页的『removeAll()』
- 第 752 页的『setField()』
- 第 754 页的『store()』
- 第 756 页的『storeCopy()』
- 第 757 页的『storeField()』
- 第 758 页的『storeNew()』

getField()

系统函数 **JavaLib.getField** 返回指定的对象或类的指定字段的值。**JavaLib.getField** 是若干 Java 访问函数的其中一个。

```

JavaLib.getField(
    identifierOrClass javeObjIdOrClass in,
    field STRING in)
returns (result anyJavaPrimitive)

```

result

结果字段是必需的，并且接收在第二个自变量中指定的字段的值。下列情况适用:

- 如果接收的值具有 `BigDecimal`、`BigInteger`、`byte`、`short`、`int`、`long`、`float` 或 `double` 类型，则结果字段必须具有数字数据类型。特征不需要与值匹配；例如，`float` 型的值可以存储在声明为不带小数位的返回变量中。有关处理溢出的详细信息，请参阅 `VGVar.handleOverflow` 和 `sysVar.overflowIndicator`。
- 如果接收的值为布尔值，则结果字段必须具有数字基本类型。值 1 表示 `true`，值 0 表示 `false`。
- 如果接收的值为字节队列，则结果字段必须具有 `HEX` 类型。有关不匹配的长度的详细信息，请参阅赋值。
- 如果接收的值为字符串或字符型，则结果字段必须具有 `CHAR`、`DBCHAR`、`MBCHAR`、`STRING` 或 `UNICODE` 类型：
 - 如果结果字段具有 `MBCHAR`、`STRING` 或 `UNICODE` 类型，则接收的值总是具有相应类型
 - 如果结果字段具有 `CHAR` 类型，则在接收的值包括与 `DBCHAR` 字符相对应的字符的情况下，可能会发生问题
 - 如果结果字段具有 `DBCHAR` 类型，则在接收的值包括与单字节字符相对应的 `Unicode` 字符的情况下，可能会发生问题

有关不匹配的长度的详细信息，请参阅赋值。

- 如果本机 Java 方法未返回值或返回 `null`，则发生错误 00001004，列示如下。

identifierOrClass

此自变量是下列其中一个实体：

- 引用对象空间中的对象的标识；或者
- Java 类的标准名称。

此自变量是字符串文字或类型为 `CHAR`、`DBCHAR`、`MBCHAR`、`STRING` 或 `UNICODE` 的变量。如果正在指定对象的标识，则该标识必须被强制类型转换为 `objID`，如稍后的示例所示。如果您打算在下一个自变量中指定静态字段，则建议在此自变量中指定一个类。

EGL 将从自变量值的开头和末尾除去单字节和双字节空格，其中，自变量值是区分大小写的。

field

要读取的字段的名称。

此自变量是字符串文字或类型为 `CHAR`、`DBCHAR`、`MBCHAR`、`STRING` 或 `UNICODE` 的变量。将从字符串的开头和末尾除去单字节和双字节空格，其中，字符串是区分大小写的。

下面是一个示例：

```
myVar = JavaLib.getField( (objId)"myID", "myField" );
```

在处理 `JavaLib.getField` 期间发生的错误会将 `sysVar.errorCode` 设置为下表中列示的值。

<code>sysVar.errorCode</code> 中的值	描述
00001000	被调用方法抛出了异常，或者作为类初始化的结果而抛出了异常

sysVar.errorCode 中的值	描述
00001001	对象为 null, 或者指定的标识不在对象空间中
00001002	具有指定名称的公用方法、字段或类不存在或无法被装入
00001004	方法返回 null、方法未返回值或者字段的值为 null
00001005	返回的值与返回变量的类型不匹配
00001007	在尝试获取关于方法或字段的信息时抛出了 SecurityException 或 IllegalAccessException; 或者尝试设置已被声明为最终字段的字段的值
00001009	必须指定标识而不是类名; 方法或字段不是静态的

相关概念

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 340 页的『赋值』

第 46 页的『BIN 和整数类型』

第 737 页的『EGL 库 JavaLib』

第 86 页的『异常处理』

『invoke()』

第 748 页的『isNull()』

第 749 页的『isObjID()』

第 750 页的『qualifiedTypeName()』

第 751 页的『remove()』

第 752 页的『removeAll()』

第 752 页的『setField()』

第 754 页的『store()』

第 756 页的『storeCopy()』

第 757 页的『storeField()』

第 758 页的『storeNew()』

invoke()

系统函数 **JavaLib.invoke** 调用本机 Java 对象或类的方法, 并有可能返回一个值。

JavaLib.invoke 是若干 Java 访问函数的其中一个。

```

JavaLib.invoke(
    identifierOrClass javaObjIdOrClass in,
    method STRING in
    {, argument anyEglPrimitive in})
returns (result anyJavaPrimitive)

```

result

返回字段 (如果有的话) 从本机 Java 方法接收值。

如果本机 Java 方法返回值, 则结果字段是可选的。

下列情况适用:

- 如果返回的值具有 `BigDecimal`、`BigInteger`、`byte`、`short`、`int`、`long`、`float` 或 `double` 类型, 则结果字段必须具有数字数据类型。特征不一定要与值匹配; 例如, 浮点值可以存储在声明为不带小数位的结果字段中。有关处理溢出的详细信息, 请参阅 `VGVar.handleOverflow` 和 `SysVar.overflowIndicator`。
- 如果返回的值为布尔值, 则结果字段必须具有数字基本类型。值 1 表示 `true`, 值 0 表示 `false`。
- 如果返回的值为字节队列, 则结果字段必须具有 `HEX` 类型。有关不匹配的长度的详细信息, 请参阅赋值。
- 如果返回的值为字符串或字符型, 则结果字段必须具有 `CHAR`、`DBCHAR`、`MBCHAR`、`STRING` 或 `UNICODE` 类型:
 - 如果结果字段具有 `MBCHAR`、`STRING` 或 `UNICODE` 类型, 则返回的值总是具有相应类型
 - 如果结果字段具有 `CHAR` 类型, 则在返回的值包括与 `DBCHAR` 字符相对应的字符的情况下, 可能会发生问题
 - 如果结果字段具有 `DBCHAR` 类型, 则在返回的值包括与单字节字符相对应的 `Unicode` 字符的情况下, 可能会发生问题

有关不匹配的长度的详细信息, 请参阅赋值。

- 如果本机 `Java` 方法未返回值或返回 `null`, 则下列情况适用:
 - 没有结果字段时, 不会发生任何错误
 - 如果结果字段存在, 运行时就会发生错误; 该错误是 00001004, 列示在后面

identifierOrClass

此自变量是下列其中一个实体:

- 引用对象空间中的对象的标识; 或者
- `Java` 类的标准名称。

此自变量是字符串文字或类型为 `CHAR`、`DBCHAR`、`MBCHAR`、`STRING` 或 `UNICODE` 的变量。如果正在指定对象的标识, 则该标识必须被强制类型转换为 `objID`, 如稍后的示例所示。EGL 将从自变量值的开头和末尾除去单字节和双字节空格, 其中, 自变量值是区分大小写的。

在创建对象的标识之前, 代码不能对该对象调用方法。后面的示例使用 `java.lang.System.out` 举例说明了这一点, 其中, `java.lang.System.out` 引用 `PrintStream` 对象。

method

要调用的方法的名称。

此自变量是字符串文字或类型为 `CHAR`、`DBCHAR`、`MBCHAR`、`STRING` 或 `UNICODE` 的变量。将从字符串的开头和末尾除去单字节和双字节空格, 其中, 字符串是区分大小写的。

argument

传递给方法的值。

可能需要进行强制类型转换, 这在 `Java` 访问 (系统字) 中有所指定。

Java 类型转换规则生效。例如，即使将 short 传递给声明为 int 的方法参数，也不会发生错误。

为避免精度降低，使用 EGL 浮点变量来表示 Java 双精度，并使用 EGL smallfloat 变量来表示 Java 浮点。使用其它 EGL 类型中的某个类型可能导致值被四舍五入。

无论方法执行什么操作，调用程序中的内存区都不会更改。

在下面的示例中，除非另有说明，否则需要进行强制类型转换 (objId)：

```
// call the constructor of the Java Date class and
// assign the new object to the identifier "date".
JavaLib.storeNew( (objId)"date", "java.util.Date");

// call the toString method of the new Date object
// and assign the output (today's date) to the chaItem.
// In the absence of the cast (objId), "date"
// refers to a class rather than an object.
chaItem = JavaLib.invoke( (objId)"date", "toString" );

// assign the standard output stream of the
// Java System class to the identifier "systemOut".
JavaLib.storeField( (objId)"systemOut", "java.lang.System", "out" );

// call the println method of the output
// stream and print today's date.
JavaLib.invoke( (objID)"systemOut", "println", chaItem );

// The use of "java.lang.System.out" as the first
// argument in the previous line would not have been
// valid, as the argument must either be a
// an identifier already in the object space or a class
// name. The argument cannot refer to a static field.
```

在处理 **JavaLib.invoke** 期间发生的错误会将 **SysVar.errorCode** 设置为下表中列示的值。

SysVar.errorCode 中的值	描述
00001000	被调用方法抛出了异常，或者作为类初始化的结果而抛出了异常
00001001	对象为 null，或者指定的标识不在对象空间中
00001002	具有指定名称的公用方法、字段或类不存在或无法被装入
00001003	EGL 基本类型与 Java 中期望的类型不匹配
00001004	方法返回 null、方法未返回值或者字段的值为 null
00001005	返回的值与返回变量的类型不匹配
00001006	未能装入强制类型转换为 null 的自变量的类
00001007	在尝试获取关于方法或字段的信息时抛出了 SecurityException 或 IllegalAccessException；或者尝试设置已被声明为最终字段的字段的值
00001009	必须指定标识而不是类名；方法或字段不是静态的

相关概念

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 340 页的『赋值』

第 46 页的『BIN 和整数类型』

第 737 页的『EGL 库 JavaLib』

第 86 页的『异常处理』

第 743 页的『getField()』

『isNull()』

第 749 页的『isObjID()』

第 750 页的『qualifiedTypeName()』

第 751 页的『remove()』

第 752 页的『removeAll()』

第 752 页的『setField()』

第 754 页的『store()』

第 756 页的『storeCopy()』

第 757 页的『storeField()』

第 758 页的『storeNew()』

第 31 页的『基本类型』

第 854 页的『overflowIndicator』

第 868 页的『handleOverflow』

isNull()

系统函数 **JavaLib.isNull** 返回一个值（1 表示 true，0 表示 false）以指示指定的标识是否引用 NULL 对象。**JavaLib.isNull** 是若干 Java 访问函数的其中一个。

```
JavaLib.isNull(identifier javaObjId in)  
returns (result INT)
```

result

一个数字字段，它接收两个值中的一个：1 表示 true，0 表示 false。使用非数字字段将导致验证时发生错误。

identifier

引用对象空间中的对象的标识。

此自变量是字符串文字或类型为 CHAR、DBCHAR、MBCHAR、STRING 或 UNICODE 的项。必须将该标识强制类型转换为 objID。EGL 将从自变量值的开头和末尾除去单字节和双字节空格，其中，自变量值是区分大小写的。

下面是一个示例：

```
// test whether an object is null  
// and process accordingly  
isNull = JavaLib.isNull( (objId)valueId );  
  
if( isNull == 1 )  
    error = 12;  
end
```

在处理 **JavaLib.isNull** 期间发生的错误会将 **SysVar.errorCode** 设置为下表中列示的值。

sysVar.errorCode 中的值	描述
00001001	指定的标识不存在于对象空间中

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 737 页的『EGL 库 JavaLib』

第 743 页的『getField()』

第 745 页的『invoke()』

『isObjID()』

第 750 页的『qualifiedTypeName()』

第 751 页的『remove()』

第 752 页的『removeAll()』

第 752 页的『setField()』

第 754 页的『store()』

第 756 页的『storeCopy()』

第 757 页的『storeField()』

第 758 页的『storeNew()』

相关任务

第 690 页的『EGL 语句和命令的语法图』

isObjID()

系统函数 **JavaLib.isObjID** 返回一个值（1 表示 true，0 表示 false）以指示指定的标识是否位于对象空间中。**JavaLib.isObjID** 是若干 Java 访问函数的其中一个。

```
JavaLib.isObjID(identifier javaObjId in)
returns (result INT)
```

result

一个数字项，它接收两个值中的一个：1 表示 true，0 表示 false。使用非数字项将导致验证时发生错误。

identifier

引用对象空间中的对象的标识。

此自变量是字符串文字或类型为 CHAR、DBCHAR、MBCHAR 或 UNICODE 的项。必须将该标识强制类型转换为 objID。EGL 将从自变量值的开头和末尾除去单字节和双字节空格，其中，自变量值是区分大小写的。

下面是一个示例：

```
// test whether an object is non-existent
// and process accordingly
isPresent = JavaLib.isObjID( (objId)valueId );

if( isPresent == 0 )
    error = 27;
end
```

JavaLib.isObjID 没有相关联的运行时错误。

相关概念

第 689 页的『EGL 函数的语法图』

第 737 页的『Java 访问函数』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 737 页的『EGL 库 JavaLib』

第 743 页的『getField()』

第 745 页的『invoke()』

第 748 页的『isNull()』

『qualifiedTypeName()』

第 751 页的『remove()』

第 752 页的『removeAll()』

第 752 页的『setField()』

第 754 页的『store()』

第 756 页的『storeCopy()』

第 757 页的『storeField()』

第 758 页的『storeNew()』

qualifiedTypeName()

系统函数 **JavaLib.qualifiedTypeName** 返回对象的类在 EGL Java 对象空间中的标准名称。**JavaLib.qualifiedTypeName** 是若干 Java 访问函数的其中一个。

```
JavaLib.qualifiedTypeName(identifier javaObjId in)  
returns (result STRING)
```

result

结果字段是必需的，并且必须具有 CHAR、MBCHAR 或 UNICODE 类型：

- 如果结果字段具有 MBCHAR 或 UNICODE 类型，则接收到的值总是合适的
- 如果结果字段具有 CHAR 类型，则在接收的值包括与 DBCHAR 字符相对应的字符的情况下，可能会发生问题

有关不匹配的长度的详细信息，请参阅赋值。

identifier

引用对象空间中的对象的标识。

此自变量是字符串文字或类型为 CHAR、DBCHAR、MBCHAR 或 UNICODE 的项。必须将该标识强制类型转换为 objId，如稍后的示例所示。EGL 将从自变量值的开头和末尾除去单字节和双字节空格，其中，自变量值是区分大小写的。

下面是一个示例：

```
myItem = JavaLib.qualifiedTypeName( (objId)"myId" );
```

在处理 **JavaLib.qualifiedTypeName** 期间发生的错误会将 **sysVar.errorCode** 设置为下表中列示的值。

sysVar.errorCode 中的值	描述
00001001	对象为 null，或者指定的标识不在对象空间中

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 737 页的『EGL 库 JavaLib』

第 743 页的『getField()』

第 745 页的『invoke()』

第 748 页的『isNull()』

第 749 页的『isObjID()』

第 750 页的『qualifiedTypeName()』

『remove()』

第 752 页的『removeAll()』

第 752 页的『setField()』

第 754 页的『store()』

第 756 页的『storeCopy()』

第 757 页的『storeField()』

第 758 页的『storeNew()』

相关任务

第 690 页的『EGL 语句和命令的语法图』

remove()

系统函数 **JavaLib.remove** 从 EGL Java 对象空间中除去指定的标识。还将除去与该标识相关的对象，但仅当该标识是唯一一个引用该对象的标识时才会这样做。如果有另一个标识引用该对象，则该对象仍存在于对象空间中，并可通过那个标识访问该对象。

JavaLib.remove 是若干 Java 访问函数的其中一个。

```
JavaLib.remove(identifier javaObjId in)
```

identifier

引用对象的标识。即使找不到该标识也不会出错。

此自变量是字符串文字或类型为 CHAR、DBCHAR、MBCHAR、STRING 或 UNICODE 的变量。必须将该标识强制类型转换为 objID，如稍后的示例所示。EGL 将从自变量值的开头和末尾除去单字节和双字节空格，其中，自变量值是区分大小写的。

下面是一个示例：

```
JavaLib.remove( (objId)myStoredObject );
```

JavaLib.remove 没有相关联的运行时错误。

注：通过调用系统函数 **JavaLib.remove** 和 **JavaLib.removeAll**，代码允许 Java 虚拟机处理 EGL Java 对象空间中的垃圾回收。如果未调用系统函数来从对象空间中除去对象，则在任何能够访问对象空间的程序的运行时期间都不会恢复内存。

相关概念

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 737 页的『EGL 库 JavaLib』

第 743 页的『getField()』

第 745 页的『invoke()』

第 748 页的『isNull()』

第 749 页的『isObjID()』

第 750 页的『qualifiedTypeName()』

『removeAll()』

『setField()』

第 754 页的『store()』

第 756 页的『storeCopy()』

第 757 页的『storeField()』

第 758 页的『storeNew()』

removeAll()

系统函数 **JavaLib.removeAll** 从 EGL Java 对象空间中除去所有标识和对象。**JavaLib.removeAll** 是若干 Java 访问函数的其中一个。

```
JavaLib.removeAll( )
```

JavaLib.removeAll 没有相关联的运行时错误。

注：通过调用系统函数 **JavaLib.remove** 和 **JavaLib.removeAll**，代码允许 Java 虚拟机处理 EGL Java 对象空间中的垃圾回收。如果未调用系统函数来从对象空间中除去对象，则在任何能够访问对象空间的程序的运行时期都不会恢复内存。

相关概念

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 737 页的『EGL 库 JavaLib』

第 743 页的『getField()』

第 745 页的『invoke()』

第 748 页的『isNull()』

第 749 页的『isObjID()』

第 750 页的『qualifiedTypeName()』

第 751 页的『remove()』

『setField()』

第 754 页的『store()』

第 756 页的『storeCopy()』

第 757 页的『storeField()』

第 758 页的『storeNew()』

setField()

系统函数 **JavaLib.setField** 设置本机 Java 对象或类中的字段的值。**JavaLib.setField** 是若干 Java 访问函数的其中一个。

```

JavaLib.setField(
  identifierOrClass javaObjId in,
  field STRING in,
  value anyEglPrimitive in)

```

identifierOrClass

此自变量是下列其中一个实体:

- 引用对象空间中的对象的标识; 或者
- Java 类的标准名称。

此自变量是字符串文字或类型为 CHAR、DBCHAR、MBCHAR、STRING 或 UNICODE 的变量。如果正在指定对象的标识, 则该标识必须被强制类型转换为 objID, 如稍后的示例所示。EGL 将从自变量值的开头和末尾除去单字节和双字节空格, 其中, 自变量值是区分大小写的。

field

要更改的字段名称。

此自变量是字符串文字或类型为 CHAR、DBCHAR、MBCHAR、STRING 或 UNICODE 的变量。将从字符串的开头和末尾除去单字节和双字节空格, 其中, 字符串是区分大小写的。

value

该值本身。

可能需要进行强制类型转换, 这在 Java 访问 (系统字) 中有所指定。

Java 类型转换规则生效。例如, 即使将 short 赋给声明为 int 的字段, 也不会发生错误。

下面是一个示例:

```

JavaLib.setField( (objID)"myId", "myField",
  (short)myNumItem );

```

在处理 **JavaLib.setField** 期间发生的错误会将 **SysVar.errorCode** 设置为下表中列出的值。

SysVar.errorCode 中的值	描述
00001000	被调用方法抛出了异常, 或者作为类初始化的结果而抛出了异常
00001001	对象为 null, 或者指定的标识不在对象空间中
00001002	具有指定名称的公用方法、字段或类不存在或无法被装入
00001003	EGL 基本类型与 Java 中期望的类型不匹配
00001007	在尝试获取关于方法或字段的信息时抛出了 SecurityException 或 IllegalAccessException; 或者尝试设置已被声明为最终字段的字段的值
00001009	必须指定标识而不是类名; 方法或字段不是静态的

相关概念

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 737 页的『EGL 库 JavaLib』

第 743 页的『getField()』

第 745 页的『invoke()』

第 748 页的『isNull()』

第 749 页的『isObjID()』

第 750 页的『qualifiedTypeName()』

第 751 页的『remove()』

第 752 页的『removeAll()』

『store()』

第 756 页的『storeCopy()』

第 757 页的『storeField()』

第 758 页的『storeNew()』

store()

系统函数 **JavaLib.store** 调用方法并将返回的对象（或 null）随指定的标识一起放到 EGL Java 对象空间中。如果该标识已存在于对象空间中，则此操作等同于下列步骤：

- 对该标识运行 **JavaLib.remove** 以除去与该标识相关的对象
- 将 **JavaLib.store** 返回的对象与目标标识相关联

如果该方法返回 Java 基本类型而不是对象，则 EGL 将存储表示该基本类型的对象；例如，如果该方法返回 int，则 EGL 将存储类型为 java.lang.Integer 的对象。

JavaLib.store 是若干 Java 访问函数的其中一个。

```
JavaLib.store(  
    storeId javaObjId in,  
    identifierOrClass javaObjId in,  
    method STRING in  
    {, argument anyEglPrimitive in} )
```

storeId

要与返回的对象一起存储的标识。

此自变量是字符串文字或类型为 CHAR、DBCHAR、MBCHAR、STRING 或 UNICODE 的变量。必须将该标识强制类型转换为 objID，如稍后的示例所示。EGL 将从自变量值的开头和末尾除去单字节和双字节空格，其中，自变量值是区分大小写的。

identifierOrClass

此自变量是下列其中一个实体：

- 引用对象空间中的对象的标识；或者
- Java 类的标准名称。

此自变量是字符串文字或类型为 CHAR、DBCHAR、MBCHAR、STRING 或 UNICODE 的项。如果正在指定对象的标识，则该标识必须被强制类型转换为 objID，如稍后的示例所示。EGL 将从自变量值的开头和末尾除去单字节和双字节空格，其中，自变量值是区分大小写的。

method

要调用的方法。

此自变量是字符串文字或类型为 CHAR、DBCHAR、MBCHAR、STRING 或 UNICODE 的变量。如果正在指定对象的标识，则该标识必须被强制类型转换为 objID，如稍后的示例所示。EGL 将从自变量值的开头和末尾除去单字节和双字节空格，其中，自变量值是区分大小写的。

argument

传递给方法的值。

可能需要进行强制类型转换，这在 Java 访问（系统字）中有所指定。

Java 类型转换规则生效。例如，即使将 short 传递给声明为 int 的方法参数，也不会发生错误。

为避免精度降低，使用 EGL 浮点项来表示 Java 双精度，并使用 EGL smallfloat 项来表示 Java 浮点。使用其它 EGL 类型中的某个类型可能导致值被四舍五入。

无论方法执行什么操作，调用程序中的内存区都不会更改。

下面是一个示例：

```
JavaLib.store( (objId)"storeId", (objId)"myId",  
              "myMethod", 36 );
```

在处理 **JavaLib.store** 期间发生的错误会将 **sysVar.errorCode** 设置为下表中列示的值。

sysVar.errorCode 中的值	描述
00001000	被调用方法抛出了异常，或者作为类初始化的结果而抛出了异常
00001001	对象为 null，或者指定的标识不在对象空间中
00001002	具有指定名称的公用方法、字段或类不存在或无法被装入
00001003	EGL 基本类型与 Java 中期望的类型不匹配
00001006	未能装入强制类型转换为 null 的自变量的类
00001007	在尝试获取关于方法或字段的信息时抛出了 SecurityException 或 IllegalAccessException；或者尝试设置已被声明为最终字段的字段的值
00001009	必须指定标识而不是类名；方法或字段不是静态的

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 737 页的『EGL 库 JavaLib』

第 743 页的『getField()』

第 745 页的『`invoke()`』
第 748 页的『`isNull()`』
第 749 页的『`isObjID()`』
第 750 页的『`qualifiedTypeName()`』
第 751 页的『`remove()`』
第 752 页的『`removeAll()`』
第 752 页的『`setField()`』
『`storeCopy()`』
第 757 页的『`storeField()`』
第 758 页的『`storeNew()`』

storeCopy()

系统函数 **JavaLib.storeCopy** 根据对象空间中的一个标识创建另一个新标识，以使两个标识引用同一对象。如果源标识不在对象空间中，则将目标标识存储为 `null`，而不会发生任何错误。如果目标标识已存在于对象空间中，则此操作等同于下列步骤：

- 对目标标识运行 **JavaLib.remove** 以除去与该标识相关的对象
- 使源对象与目标标识相关

JavaLib.storeCopy 是若干 Java 访问函数的其中一个。

```
JavaLib.storeCopy(  
    sourceId javaObjId in,  
    targetId javaObjId in)
```

sourceId

引用对象空间中的对象或者引用 `null` 的标识。

此自变量是字符串文字或类型为 `CHAR`、`DBCHAR`、`MBCHAR`、`STRING` 或 `UNICODE` 的变量。必须将该标识强制类型转换为 `objId`，如稍后的示例所示。EGL 将从自变量值的开头和末尾除去单字节和双字节空格，其中，自变量值是区分大小写的。

targetId

新标识符，它引用同一个对象。

此自变量是字符串文字或类型为 `CHAR`、`DBCHAR`、`MBCHAR`、`STRING` 或 `UNICODE` 的项。必须将该标识强制类型转换为 `objID`，如稍后的示例所示。EGL 将从自变量值的开头和末尾除去单字节和双字节空格，其中，自变量值是区分大小写的。

下面是一个示例：

```
JavaLib.storeCopy( (objId)"sourceId", (objId)"targetId" );
```

JavaLib.storeCopy 没有相关联的运行时错误。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 737 页的『EGL 库 JavaLib』
第 743 页的『`getField()`』
第 745 页的『`invoke()`』
第 748 页的『`isNull()`』

第 749 页的 『 isObjID() 』
第 750 页的 『 qualifiedTypeName() 』
第 751 页的 『 remove() 』
第 752 页的 『 removeAll() 』
第 752 页的 『 setField() 』
第 754 页的 『 store() 』
『 storeField() 』
第 758 页的 『 storeNew() 』

storeField()

系统函数 **JavaLib.storeField** 将类字段或对象字段的值放到 EGL Java 对象空间中。如果用来存储该对象的标识已存在于对象空间中，则此操作等同于下列步骤：

- 对该标识运行 **JavaLib.remove** 以除去与该标识相关的对象
- 使新对象与该标识相关

如果类或对象字段包含 Java 基本类型而不是对象，则 EGL 将存储表示该基本类型的对象；例如，如果字段包含一个 `int`，则 EGL 将存储类型为 `java.lang.Integer` 的对象。

```
JavaLib.storeField(  
    storeId javaObjId in,  
    identifierOrClass javaObjIdOrClass in,  
    field STRING in)
```

storeId

要与对象存储在一起的标识。

此自变量是字符串文字或类型为 `CHAR`、`DBCHAR`、`MBCHAR` 或 `UNICODE` 的项。必须将该标识强制类型转换为 `objID`，如稍后的示例所示。EGL 将从自变量值的开头和末尾除去单字节和双字节空格，其中，自变量值是区分大小写的。

identifierOrClass

此自变量是下列其中一个实体：

- 引用对象空间中的对象的标识；或者
- Java 类的标准名称。

此自变量是字符串文字或类型为 `CHAR`、`DBCHAR`、`MBCHAR` 或 `UNICODE` 的项。如果正在指定对象的标识，则该标识必须被强制类型转换为 `objID`，如稍后的示例所示。如果您打算在下一个自变量中指定静态字段，则建议在此自变量中指定一个类。

EGL 将从自变量值的开头和末尾除去单字节和双字节空格，其中，自变量值是区分大小写的。

field

引用对象的字段的名称。

此自变量是字符串文字或类型为 `CHAR`、`DBCHAR`、`MBCHAR` 或 `UNICODE` 的项。将从字符串的开头和末尾除去单字节和双字节空格，其中，字符串是区分大小写的。

下面是一个示例：

```
JavaLib.storeField( (objId)"myStoreId",  
    (objId)"myId", "myField");
```

在处理 **JavaLib.storeField** 期间发生的错误会将 **sysVar.errorCode** 设置为下表中列示的值。

sysVar.errorCode 中的值	描述
00001000	被调用方法抛出了异常，或者作为类初始化的结果而抛出了异常
00001001	对象为 null，或者指定的标识不在对象空间中
00001002	具有指定名称的公用方法、字段或类不存在或无法被装入
00001007	在尝试获取关于方法或字段的信息时抛出了 <code>SecurityException</code> 或 <code>IllegalAccessException</code> ；或者尝试设置已被声明为最终字段的字段的值
00001009	必须指定标识而不是类名；方法或字段不是静态的

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 737 页的『EGL 库 JavaLib』

第 743 页的『getField()』

第 745 页的『invoke()』

第 748 页的『isNull()』

第 749 页的『isObjID()』

第 750 页的『qualifiedTypeName()』

第 751 页的『remove()』

第 752 页的『removeAll()』

第 752 页的『setField()』

第 754 页的『store()』

第 756 页的『storeCopy()』

『storeNew()』

相关任务

第 690 页的『EGL 语句和命令的语法图』

storeNew()

系统函数 **JavaLib.storeNew** 调用类的构造函数并将新对象放到 EGL Java 对象空间中。如果该标识已存在于对象空间中，则此操作等同于下列步骤：

- 对该标识运行 **JavaLib.remove** 以除去先前与该标识相关联的对象
- 使新对象与该标识相关

JavaLib.storeNew 是若干 Java 访问函数的其中一个。

```
JavaLib.storeNew(  
    storeId javaObjId in,  
    class STRING in  
    {, argument anyEglPrimitive in})
```

storeId

要与新对象一起存储的标识。

此自变量是字符串文字或类型为 CHAR、DBCHAR、MBCHAR 或 UNICODE 的项。必须将该标识强制类型转换为 objID，如稍后的示例所示。EGL 将从自变量值的开头和末尾除去单字节和双字节空格，其中，自变量值是区分大小写的。

class

Java 类的标准名称。

此自变量是字符串文字或类型为 CHAR、DBCHAR、MBCHAR 或 UNICODE 的项。EGL 将从自变量值的开头和末尾除去单字节和双字节空格，其中，自变量值是区分大小写的。

argument

传递给构造函数的值。

可能需要进行强制类型转换，这在 *Java 访问*（系统字）中有所指定。

Java 类型转换规则生效。例如，即使将 short 传递给声明为 int 的构造函数参数，也不会发生错误。

为避免精度降低，使用 EGL 浮点项来表示 Java 双精度，并使用 EGL smallfloat 项来表示 Java 浮点。使用其它 EGL 类型中的某个类型可能导致值被四舍五入。

无论构造函数执行什么操作，调用程序中的内存区都不会更改。

下面是一个示例：

```
JavaLib.storeNew( (objId)"storeId", "myClass", 36 );
```

在处理 **JavaLib.storeNew** 期间发生的错误会将 **sysVar.errorCode** 设置为下表中列示的值。

sysVar.errorCode 中的值	描述
00001000	被调用方法抛出了异常，或者作为类初始化的结果而抛出了异常
00001001	对象为 null，或者指定的标识不在对象空间中
00001002	具有指定名称的公用方法、字段或类不存在或无法被装入
00001003	EGL 基本类型与 Java 中期望的类型不匹配
00001006	未能装入强制类型转换为 null 的自变量的类
00001007	在尝试获取关于方法或字段的信息时抛出了 SecurityException 或 IllegalAccessException；或者尝试设置已被声明为最终字段的字段的值
00001008	不能调用构造函数；类名引用接口或抽象类

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 737 页的『EGL 库 JavaLib』

第 743 页的『getField()』

第 745 页的『invoke()』

第 748 页的『isNull()』

第 749 页的『isObjID()』

- 第 750 页的『qualifiedTypeName()』
- 第 751 页的『remove()』
- 第 752 页的『removeAll()』
- 第 752 页的『setField()』
- 第 754 页的『store()』
- 第 756 页的『storeCopy()』
- 第 757 页的『storeField()』

相关任务

- 第 690 页的『EGL 语句和命令的语法图』

EGL 库 LobLib

下表列出库 LobLib 中的函数。

系统函数 / 调用	描述
<code>attachBlobToFile(blobVariable, fileName)</code>	将类型为 BLOB 的变量引用的数据复制到指定文件中。
<code>attachBlobToTempFile(blobVariable)</code>	将类型为 BLOB 的变量引用的数据复制到唯一的临时系统文件中。
<code>attachClobToFile(clobVariable, fileName)</code>	将类型为 CLOB 的变量引用的数据复制到指定文件中。
<code>attachClobToTempFile(clobVariable)</code>	将类型为 CLOB 的变量引用的数据复制到唯一的临时系统文件中。
<code>freeBlob(blobVariable)</code>	释放类型为 BLOB 的变量使用的资源。
<code>freeClob(clobVariable)</code>	释放类型为 CLOB 的变量使用的资源。
<code>result = getBlobLen(blobVariable)</code>	返回类型为 BLOB 的变量引用的值中的字节数。
<code>result = getClobLen(clobVariable)</code>	返回类型为 CLOB 的变量引用的字符数。
<code>result = getStrFromClob(clobVariable)</code>	返回对应于类型为 CLOB 的变量引用的值的字符串。
<code>result = getSubStrFromClob(clobVariable, pos, length)</code>	返回类型为 CLOB 的变量引用的值中的子串。
<code>loadBlobFromFile(blobVariable, fileName)</code>	将指定文件中的数据复制至类型为 BLOB 的变量引用的内存区。
<code>loadClobFromFile(blobVariable, fileName)</code>	将指定文件中的数据复制至类型为 CLOB 的变量引用的内存区。
<code>setClobFromString(clobVariable, str)</code>	将字符串复制至类型为 CLOB 的变量引用的内存区。
<code>setClobFromStringAtPosition(clobVariable, pos, str)</code>	将字符串复制至类型为 CLOB 的变量引用的内存区，从内存区中的指定位置开始。
<code>truncateBlob(blobVariable, length)</code>	截断类型为 BLOB 的变量引用的值。
<code>truncateClob(clobVariable, length)</code>	截断类型为 CLOB 的变量引用的值。
<code>updateBlobToFile(blobVariable, fileName)</code>	将类型为 BLOB 的变量引用的数据复制到指定文件中。
<code>updateClobToFile(blobVariable, fileName)</code>	将类型为 CLOB 的变量引用的数据复制到指定文件中。

attachBlobToFile()

系统函数 **LobLib.attachBlobToFile** 将类型为 BLOB 的变量引用的数据复制到指定文件中。

```
LobLib.attachBlobToFile(  
    blobVariable BLOB inOut,  
    fileName STRING in)
```

blobVariable

类型为 BLOB 的变量。

fileName

文件的名称。该名称是标准名称或是相对于从中调用该程序的目录的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 45 页的『BLOB』

第 760 页的『EGL 库 LobLib』

attachBlobToTempFile()

系统函数 **LobLib.attachBlobToTempFile** 将类型为 BLOB 的变量引用的数据复制到唯一的临时系统文件中。此函数将运行时使用的内存量降至最低。

```
LobLib.attachBlobToTempFile(blobVariable BLOB in)
```

blobVariable

类型为 BLOB 的变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 45 页的『BLOB』

第 760 页的『EGL 库 LobLib』

attachClobToFile()

系统函数 **LobLib.attachClobToFile** 将类型为 CLOB 的变量引用的数据复制到指定文件中。

```
LobLib.attachClobToFile(  
    clobVariable CLOB inOut,  
    fileName STRING in)
```

clobVariable

类型为 CLOB 的变量。

fileName

文件的名称。该名称是标准名称或是相对于从中调用该程序的目录的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 44 页的『CLOB』
第 760 页的『EGL 库 LobLib』

attachClobToTempFile()

系统函数 **LobLib.attachClobToTempFile** 将类型为 CLOB 的变量引用的数据复制到唯一的临时系统文件中。此函数将运行时使用的内存量降至最低。

```
LobLib.attachClobToTempFile(clobVariable CLOB in)
```

clobVariable

类型为 CLOB 的变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 44 页的『CLOB』
第 760 页的『EGL 库 LobLib』

freeBlob()

系统函数 **LobLib.freeBlob** 释放类型为 BLOB 的变量使用的所有资源。

```
LobLib.freeBlob(blobVariable BLOB inOut)
```

blobVariable

类型为 BLOB 的变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 45 页的『BLOB』
第 760 页的『EGL 库 LobLib』

freeClob()

系统函数 **LobLib.freeClob** 释放类型为 CLOB 的变量使用的资源。

```
LobLib.freeClob(clobVariable CLOB inOut)
```

clobVariable

类型为 CLOB 的变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 44 页的『CLOB』
第 760 页的『EGL 库 LobLib』

getBlobLen()

系统函数 **LobLib.getBlobLen** 返回类型为 BLOB 的变量引用的值中的字节数。

LobLib.getBlobLen(*blobVariable* **BLOB** in)
returns (*result* **BIGINT**)

result

字节数。

blobVariable

类型为 BLOB 的变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 45 页的『BLOB』

第 760 页的『EGL 库 LobLib』

getClobLen()

系统函数 **LobLib.getClobLen** 返回类型为 CLOB 的变量引用的字符数。

LobLib.getClobLen(*clobVariable* **CLOB** in)
returns (*result* **BIGINT**)

result

字符数。

clobVariable

类型为 CLOB 的变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 44 页的『CLOB』

第 760 页的『EGL 库 LobLib』

getStrFromClob()

系统函数 **LobLib.getStrFromClob** 返回对应于类型为 CLOB 的变量引用的值的字符串。

LobLib.getStrFromClob(*clobVariable* **CLOB** in)
returns (*result* **STRING**)

result

返回的字符串。

clobVariable

类型为 CLOB 的变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 44 页的『CLOB』

第 760 页的『EGL 库 LobLib』

getSubStrFromClob()

系统函数 **LobLib.getSubStrFromClob** 返回类型为 CLOB 的变量引用的值中的子串。

```
LobLib.getSubStrFromClob(  
  clobVariable CLOB in,  
  pos BIGINT in,  
  length BIGINT in)  
returns (result STRING)"
```

result

类型为 STRING 的值。

clobVariable

类型为 CLOB 的变量。

pos

标识子串开头的字符的数字位置。CLOB 变量中的第一个字符位于位置 1。

length

标识子串中的字符数。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 44 页的『CLOB』

第 760 页的『EGL 库 LobLib』

loadBlobFromFile()

系统函数 **LobLib.loadBlobFromFile** 将指定文件中的数据复制至类型为 BLOB 的变量引用的内存区。

```
LobLib.loadBlobFromFile(  
  blobVariable BLOB inOut,  
  fileName STRING in)
```

blobVariable

类型为 BLOB 的变量。

fileName

文件的名称。该名称是标准名称或是相对于从中调用该程序的目录的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 45 页的『BLOB』

第 760 页的『EGL 库 LobLib』

loadClobFromFile()

系统函数 **LobLib.loadClobFromFile** 将指定文件中的数据复制至类型为 CLOB 的变量引用的内存区。

```
LobLib.loadClobFromFile(  
  clobVariable CLOB inOut,  
  fileName STRING in)
```

clobVariable

类型为 CLOB 的变量。

fileName

文件的名称。该名称是标准名称或是相对于从中调用该程序的目录的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 44 页的『CLOB』

第 760 页的『EGL 库 LobLib』

setClobFromString()

系统函数 **LobLib.setClobFromString** 将字符串复制至类型为 CLOB 的变量引用的内存区。

```
LobLib.setClobFromString(  
  clobVariable CLOB inOut,  
  str STRING in)
```

clobVariable

类型为 CLOB 的变量。

str 要复制的字符串。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 44 页的『CLOB』

第 760 页的『EGL 库 LobLib』

setClobFromStringAtPosition()

系统函数 **LobLib.setClobFromStringAtPosition** 将字符串复制至类型为 CLOB 的变量引用的内存区，从内存区中的指定位置开始。

```
LobLib.setClobFromStringAtPosition(  
  clobVariable CLOB inOut,  
  pos BIGINT in  
  str STRING in)
```

clobVariable

类型为 CLOB 的变量。

pos

clobVariable 引用的值中的字符位置。CLOB 变量中的第一个字符位于位置 1。

str 要复制的字符串。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 44 页的『CLOB』

第 760 页的『EGL 库 LobLib』

truncateBlob()

系统函数 **LobLib.truncateBlob** 截断类型为 BLOB 的变量引用的值。

```
LobLib.truncateBlob(  
  blobVariable BLOB inOut,  
  length BIGINT in)
```

blobVariable

类型为 BLOB 的变量。

length

输出中的字节数。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 45 页的『BLOB』

第 760 页的『EGL 库 LobLib』

truncateClob()

系统函数 **LobLib.truncateClob** 截断类型为 CLOB 的变量引用的值。

```
LobLib.truncateClob(  
  clobVariable CLOB inOut,  
  length BIGINT in)
```

clobVariable

类型为 CLOB 的变量。

length

输出中的字节（不是字符）数。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 44 页的『CLOB』

第 760 页的『EGL 库 LobLib』

updateBlobToFile()

系统函数 **LobLib.updateBlobToFile** 将类型为 BLOB 的变量引用的数据复制到指定文件中。如果该文件存在，则该函数先擦除该文件的内容；否则，该函数将创建该文件。

```
LobLib.updateBlobToFile(  
  blobVariable BLOB inOut,  
  fileName STRING in)
```

blobVariable

类型为 BLOB 的变量。

fileName

文件的名称。该名称是标准名称或是相对于从中调用该程序的目录的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 45 页的『BLOB』

第 760 页的『EGL 库 LobLib』

updateClobToFile()

系统函数 **LobLib.updateClobToFile** 将类型为 CLOB 的变量引用的数据复制到指定文件中。如果该文件存在，则该函数先擦除该文件的内容；否则，该函数将创建该文件。

```
LobLib.updateClobToFile(  
    clobVariable CLOB inOut,  
    fileName STRING in)
```

clobVariable

类型为 CLOB 的变量。

fileName

文件的名称。该名称是标准名称或是相对于从中调用该程序的目录的名称。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 44 页的『CLOB』

第 760 页的『EGL 库 LobLib』

EGL 库 MathLib

下表列出系统库 MathLib 中的函数。

注： 字段 *numericField* 属于类型 BIGINT、BIN、DECIMAL、HEX、INT、NUM、NUMC、PACF、SMALLINT、FLOAT 或 SMALLFLOAT。

类型为 HEX 的字段（长度为 8）被假定为具有单精度，即运行时环境本地的 4 字节浮点数；类型为 HEX 的字段（长度为 16）被假定为双精度，即运行时环境本地的 8 字节浮点数。

系统函数 / 调用	描述
<i>result</i> = abs (<i>numericField</i>)	返回 <i>numericField</i> 的绝对值
<i>result</i> = acos (<i>numericField</i>)	返回 <i>numericField</i> 的反余弦
<i>result</i> = asin (<i>numericField</i>)	返回 <i>numericField</i> 的反正弦
<i>result</i> = atan (<i>numericField</i>)	返回 <i>numericField</i> 的反正切
<i>result</i> = atan2 (<i>numericField1</i> , <i>numericField2</i>)	通过使用两个自变量的符号确定返回值的象限，计算 <i>numericField1</i> / <i>numericField2</i> 的反正切的主值
<i>result</i> = ceiling (<i>numericField</i>)	返回不小于 <i>numericField</i> 的最小整数
<i>result</i> = compareNum (<i>numericField1</i> , <i>numericField2</i>)	返回结果（-1、0 或 1），它指示 <i>numericField1</i> 是小于、等于还是大于 <i>numericField2</i>

系统函数 / 调用	描述
<code>result = cos (numericField)</code>	返回 <code>numericField</code> 的余弦
<code>result = cosh (numericField)</code>	返回 <code>numericField</code> 的双曲余弦
<code>result = exp (numericField)</code>	返回 <code>numericField</code> 的指数值
<code>result = floatingAssign (numericField)</code>	将 <code>numericField</code> 作为双精度浮点数返回
<code>result = floatingDifference (numericField1, numericField2)</code>	返回 <code>numericField1</code> 与 <code>numericField2</code> 的差
<code>result = floatingMod (numericField1, numericField2)</code>	计算 <code>numericField1</code> 除以 <code>numericField2</code> 的浮点余数, 结果与 <code>numericField1</code> 具有相同的符号
<code>result = floatingProduct (numericField1, numericField2)</code>	返回 <code>numericField1</code> 与 <code>numericField2</code> 的乘积
<code>result = floatingQuotient (numericField1, numericField2)</code>	返回 <code>numericField1</code> 除以 <code>numericField2</code> 的商
<code>result = floatingSum (numericField1, numericField2)</code>	返回 <code>numericField1</code> 与 <code>numericField2</code> 的和
<code>result = floor (numericField)</code>	返回不大于 <code>numericField</code> 的最大整数
<code>result = frexp (numericField, integer)</code>	将数字分割为 .5 到 1 范围内的规范化分数 (返回的值) 以及返回的 <code>integer</code> 的 2 次幂
<code>result = ldexp (numericField, integer)</code>	返回 <code>numericField</code> 乘以 2 的 <code>integer</code> 次幂
<code>result = log (numericField)</code>	返回 <code>numericField</code> 的自然对数
<code>result = log10 (numericField)</code>	返回 <code>numericField</code> 的底数为 10 的对数
<code>result = maximum (numericField1, numericField2)</code>	返回 <code>numericField1</code> 和 <code>numericField2</code> 中较大的那一个
<code>result = minimum (numericField1, numericField2)</code>	返回 <code>numericField1</code> 和 <code>numericField2</code> 中较小的那一个
<code>result = modf (numericField1, numericField2)</code>	将 <code>numericField1</code> 分割为整数部分和小数部分, 这两部分都与 <code>numericField1</code> 具有相同的符号; 将整数部分放在 <code>numericField2</code> 中; 返回小数部分
<code>result = pow (numericField1, numericField2)</code>	返回 <code>numericField1</code> 的 <code>numericField2</code> 次幂
<code>result = precision (numericField)</code>	返回 <code>numericField</code> 的最大精度 (以小数位数计)
<code>result = round (numericField[, integer])</code> <code>result = mathLib.round(numericExpression)</code>	将数字或表达式四舍五入到最接近的值 (例如, 四舍五入到最接近的千位) 并返回结果
<code>result = sin (numericField)</code>	返回 <code>numericField</code> 的正弦
<code>result = sinh (numericField)</code>	返回 <code>numericField</code> 的双曲正弦
<code>result = sqrt (numericField)</code>	如果 <code>numericField</code> 大于等于零, 则返回 <code>numericField</code> 的平方根
<code>result = stringAsDecimal (numberAsText)</code>	接受字符值 (如 "98.6") 并返回类型为 DECIMAL 的等效值
<code>result = stringAsFloat (numberAsText)</code>	接受字符值 (如 "98.6") 并返回类型为 FLOAT 的等效值
<code>result = stringAsInt (numberAsText)</code>	接受字符值 (如 "98.6") 并返回类型为 BIGINT 的等效值

系统函数 / 调用	描述
<code>result = tan (numericField)</code>	返回 <i>numericField</i> 的正切
<code>result = tanh (numericField)</code>	返回 <i>numericField</i> 的双曲正切

abs()

系统函数 **MathLib.abs** 返回数字的绝对值。

```
MathLib.abs(numericField mathLibNumber in)
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。*numericItem* 的绝对值被转换为具有 *result* 的格式并在 *result* 中返回。

numericField

任何数字项或 HEX 项，如数学（系统字）中所述。

MathLib.abs 能够在每个目标系统上起作用。对于 Java 程序，EGL 使用 Java StrictMath 类中的其中一个 `abs()` 方法，因此运行时行为对每个 Java 虚拟机都是相同的。

示例:

```
myItem = -5;
result = MathLib.abs(myItem); // result = 5
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

acos()

系统函数 **MathLib.acos** 返回自变量的反余弦，以弧度计。

```
MathLib.acos(numericField mathLibNumber in)
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。返回的值（介于 0.0 与 pi 之间）以弧度计并被转换为具有 *result* 的格式。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。在进行计算之前，该项被转换为双精度浮点。如果值并非介于 -1 与 1 之间，则发生错误。

MathLib.acos 能够在每个目标系统上起作用。对于 Java 程序，EGL 使用 Java StrictMath 类中的 `acos()` 方法，因此运行时行为对每个 Java 虚拟机都相同。

示例:

```
result = MathLib.acos(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

asin()

系统函数 **MathLib.asin** 返回一个数字的反正弦，该数字位于 -1 到 1 的范围内。结果是以弧度计的，并且位于 $-\pi/2$ 到 $\pi/2$ 的范围内。

```
MathLib.asin(numericField mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。MathLib.asin 函数返回的值被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。在调用 mathLib.asin 函数之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.asin(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

atan()

系统函数 **MathLib.atan** 返回数字的反正切。结果是以弧度计的，并且位于 $-\pi/2$ 到 $\pi/2$ 的范围内。

```
MathLib.atan(numericField mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。MathLib.atan 返回的值被转换为具有 *result* 的格式。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。在调用 **MathLib.atan** 之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.atan(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

atan2()

系统函数 **MathLib.atan2** 计算 y/x 的反正切的主值，并使用两个自变量的符号来确定返回值的象限。结果是以弧度计的，并且位于 $-\pi$ 到 π 的范围内。

```
MathLib.atan2(  
    numericField1 mathLibNumber in,  
    numericField2 mathLibNumber  $\overline{in}$ )  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。**MathLib.atan2** 返回的值被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField1

任何数字或 HEX 项，如数学（系统字）中所述。在调用 **MathLib.atan2** 之前，该项被转换为双精度浮点。*numericField1* 为 y 值。

numericField2

任何数字或 HEX 项，如数学（系统字）中所述。在调用 **MathLib.atan2** 之前，该项被转换为双精度浮点。*numericField2* 为 x 值。

示例:

```
myItemY = 1;  
myItemX = 5;  
  
// returns pi/2  
result = MathLib.atan2(myItemY, myItemX);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

ceiling()

系统函数 **MathLib.ceiling** 返回不小于指定数字的最小整数。

```
MathLib.ceiling(numericField mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。不小于 *numericItem* 的最小整数被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。

示例:

```
myItem = 4.5;  
result = MathLib.ceiling(myItem); // result = 5
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

compareNum()

系统函数 **MathLib.compareNum** 返回一个结果（-1、0 或 1），该结果指示两个数中的第一个数是小于、等于还是大于第二个数。

```
MathLib.compareNum(  
    numericField1 mathLibNumber in,  
    numericField2 mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

被定义为具有 INT 类型或者具有以下等效类型的项: BIN 类型, 长度为 9 并且不带小数位。此项接收下列其中一个值:

-1 *numericField1* 小于 *numericField2*。

0 *numericField1* 等于 *numericField2*。

1 *numericField1* 大于 *numericField2*。

numericField1

任何数字或 HEX 项, 如数学 (系统字) 中所述。

numericField2

任何数字或 HEX 项, 如数学 (系统字) 中所述。

示例:

```
myItem01 = 4  
myItem02 = 7  
  
result = MathLib.compareNum(myItem01,myItem02);  
  
// result = -1
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

cos()

系统函数 **MathLib.cos** 返回数字的余弦。返回的值位于 -1 到 1 的范围内。

```
MathLib.cos(numericField mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项, 如数学 (系统字) 中所述。 **MathLib.cos** 返回的值被转换为具有 *result* 的格式, 并在 *result* 中返回。

numericField

任何数字或 HEX 项, 如数学 (系统字) 中所述。在调用 **MathLib.cos** 之前, 该项被转换为双精度浮点。

示例:

```
result = MathLib.cos(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

cosh()

系统函数 **MathLib.cosh** 返回数字的双曲余弦。

```
MathLib.cosh(numericField mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。mathLib.cosh 返回的值被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。在调用 mathLib.cosh 之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.cosh(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

exp()

系统函数 **MathLib.exp** 返回 e 的数字次幂。

```
MathLib.exp(numericField mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如 *MathLib* 中所述。**MathLib.exp** 返回的值被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField

任何数字或 HEX 项，如 *MathLib* 中所述。在调用 **MathLib.exp** 之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.exp(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

floatingAssign()

系统函数 **MathLib.floatingAssign** 将 *numericItem* 作为双精度浮点数返回。此函数将 BIN、DECIMAL、NUM、NUMC 或 PACKF 项的值赋给作为 HEX 项定义的浮点数，反之亦然。

```
MathLib.floatingAssign(numericField mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。浮点数被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。该项在被赋给结果之前，它被转换为双精度浮点。

示例:

```
result = MathLib.floatingAssign(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

floatingDifference()

系统函数 **MathLib.floatingDifference** 将两个数中的第一个数减去第二个数并返回差。此函数是使用双精度浮点算术实现的。

```
MathLib.floatingDifference(  
    numericField1 mathLibNumber in,  
    numericField2 mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。差被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField1

任何数字或 HEX 项，如数学（系统字）中所述。在计算差之前，该项被转换为双精度浮点。

numericField2

任何数字或 HEX 项，如数学（系统字）中所述。在计算差之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.floatingDifference(myItem01,myItem02);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

floatingMod()

系统函数 **MathLib.floatingMod** 返回一个数除以另一个数的浮点余数。结果与分子具有相同的符号。如果分母等于零，则发生域异常。

```
MathLib.floatingMod(  
    numericField1 mathLibNumber in,  
    numericField2 mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。浮点余数被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField1

任何数字或 HEX 项，如数学（系统字）中所述。在计算 *result* 之前，该项被转换为双精度浮点。

numericField2

任何数字或 HEX 项，如数学（系统字）中所述。在计算 *result* 之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.floatingMod(myItem01,myItem02);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

floatingProduct()

系统函数 **MathLib.floatingProduct** 返回两个数的乘积。此函数是使用双精度浮点算术实现的。

```
MathLib.floatingProduct(  
    numericField1 mathLibNumber in,  
    numericField2 mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。乘积被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField1

任何数字或 HEX 项，如数学（系统字）中所述。在计算 *result* 之前，该项被转换为双精度浮点。

numericField2

任何数字或 HEX 项，如数学（系统字）中所述。在计算 *result* 之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.floatingProduct(myItem01,myItem02);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

floatingQuotient()

系统函数 **MathLib.floatingQuotient** 返回一个数除以另一个数的商。如果分母等于零，则发生域异常。此函数是使用双精度浮点算术实现的。

```
MathLib.floatingQuotient(  
    numericField1 mathLibNumber in,  
    numericField2 mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。商被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField1

任何数字或 HEX 项，如数学（系统字）中所述。在计算商之前，该项被转换为双精度浮点。

numericField1

任何数字或 HEX 项，如数学（系统字）中所述。在计算商之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.floatingQuotient(myItem01,myItem02);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

floatingSum()

系统函数 **MathLib.floatingSum** 返回两个数的和。此函数是使用双精度浮点算术实现的。

```
MathLib.floatingSum(  
    numericField1 mathLibNumber in,  
    numericField2 mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。和数被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField1

任何数字或 HEX 项，如数学（系统字）中所述。在计算和之前，该项被转换为双精度浮点。

numericField1

任何数字或 HEX 项，如数学（系统字）中所述。在计算和之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.floatingSum(myItem01,myItem02);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

floor()

系统函数 **MathLib.floor** 返回不大于指定数字的最大整数。

```
MathLib.floor(numericField mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。不大于 *numericField* 的最大整数被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。

示例:

```
myItem = 4.6;  
result = MathLib.floor(myItem); // result = 4
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

frexp()

系统函数 **MathLib.frexp** 将数字分割为 .5 到 1 范围内的标准化分数（作为 *result* 返回）以及返回的 *exponent* 的 2 次幂。

```
MathLib.frexp(  
  numericField mathLibNumber in,  
  exponent mathLibInteger inOut)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。浮点分数被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。在计算 *result* 之前，该项被转换为双精度浮点。

exponent

被定义为具有 INT 类型或者具有以下等效类型的项：BIN 类型，长度为 9 并且不带小数位。

示例:

```
result = MathLib.frexp(myItem,myInteger);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

Ldexp()

系统函数 **MathLib.Ldexp** 返回指定的数乘以以下值的值：2 的 *exponent* 次幂。

```
MathLib.Ldexp(  
    numericField mathLibNumber in,  
    exponent mathLibInteger in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。计算得到的值被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。在计算 *result* 之前，该项被转换为双精度浮点。

exponent

被定义为具有 INT 类型或者具有以下等效类型的项: BIN 类型，长度为 9 并且不带小数位。

示例:

```
result = MathLib.Ldexp(myItem,myInteger);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

log()

系统函数 **MathLib.log** 返回数字的自然对数。

```
MathLib.log(numericField mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。mathLib.log 函数返回的值被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。在计算 *result* 之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.log(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

log10()

系统函数 **MathLib.log10** 返回一个数的以 10 为底的对数。

```
MathLib.log10(numericField mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。log10 函数返回的值被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。在计算 *result* 之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.log10(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

maximum()

系统函数 **MathLib.maximum** 返回两个数中的较大的数。

```
MathLib.maximum(  
    numericField1 mathLibNumber in,  
    numericField2 mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。两个数中的较大的数被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField1

任何数字或 HEX 项，如数学（系统字）中所述。

numericField2

任何数字或 HEX 项，如数学（系统字）中所述。

示例:

```
result = MathLib.maximum(myItem01,myItem02);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

minimum()

系统函数 **MathLib.minimum** 返回两个数中的较小的数。

```
MathLib.minimum(  
    numericField1 mathLibNumber in,  
    numericField2 mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。两个数中的较小的数被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField1

任何数字或 HEX 项，如数学（系统字）中所述。

numericField2

任何数字或 HEX 项，如数学（系统字）中所述。

示例:

```
result = MathLib.minimum(myItem01,myItem02);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

modf()

系统函数 **MathLib.modf** 将一个数字分割为整数部分和小数部分，这两个部分都与该数字具有相同的符号。小数部分在 *result* 中返回，整数部分在 *numericField2* 中返回。

```
MathLib.modf(  
    numericField1 mathLibNumber in,  
    numericField2 mathLibNumber inOut)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。*numericField1* 的小数部分被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField1

任何数字或 HEX 项，如数学（系统字）中所述。

numericField2

任何数字或 HEX 项，如数学（系统字）中所述。*numericField1* 的整数部分被转换为具有 *numericField2* 的格式，并在 *numericField2* 中返回。

示例:

```
result = MathLib.modf(myItem01,myItem02);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

pow()

系统函数 **MathLib.pow** 返回一个数字的第二个数字次幂。在 $\text{pow}(x,y)$ 中，如果 x 的值是负数而 y 是非整数，或者 x 的值是 0.0 而 y 是负数，则发生域异常。

```
MathLib.pow(  
    numericField1 mathLibNumber in,  
    numericField2 mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。**mathLib.pow** 函数的结果被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField1

任何数字或 HEX 项，如数学（系统字）中所述。在计算 *result* 之前，该项被转换为双精度浮点。

numericField2

任何数字或 HEX 项，如数学（系统字）中所述。在计算 *result* 之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.pow(myItem01,myItem02);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

precision()

系统函数 **MathLib.precision** 返回数字的最大精度（以小数位数计）。对于浮点数（对于标准精度浮点数，是 8 位的 HEX，对于双精度浮点数，是 16 位的 HEX），精度是对于运行程序的系统，可以在数字中表示的最大小数位数。

```
MathLib.precision(numericField mathLibNumber in)  
returns (result INT)
```

result

用于接收 *numericItem* 的精度项。*result* 项被定义为具有 INT 类型，或者具有以下等效类型：BIN 类型，长度为 9 并且不带小数位。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。

示例:

```
result = MathLib.precision(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

round()

系统函数 **MathLib.round** 将数字或表达式四舍五入到最接近的值（例如，四舍五入到最接近的千位）并返回结果。

```
MathLib.round(  
  numericField mathLibNumber in  
  [, powerOf10 mathLibInteger in  
  ])  
returns (result mathLibTypeDependentResult)
```

```
MathLib.round(numericExpression anyNumericExpression in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。四舍五入运算生成的值被转换为具有 *result* 的格式，并在 *result* 中返回。

由于四舍五入是按以下方式进行的，所以在这种情况下支持的最大长度是 31 而不是 32

- 在比结果位的精度高 1 的精度位置，对 *result* 中的位加 5
- 将结果截断

如果在计算中使用了超过 31 位，并且如果 EGL 在开发时无法确定违例，则在运行时会发生数字溢出。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。

numericExpression

数字表达式，而不是简单的数字项。如果指定了运算符，则不能对 *integer* 指定值。

不能将 **MathLib.round** 与余数运算符 (%) 配合使用。

powerOf10

一个整数，它确定将数字四舍五入到的值：

- 如果 *integer* 是正数，则将数字四舍五入到最接近于 10 的 *powerOf10* 次幂的值。例如，如果 *integer* 是 3，则将数字四舍五入到最接近的千位。
- 如果 *integer* 是零或负数，情况亦如此；在那种情况下，将把数字四舍五入到指定的小数位数。

如果未指定 *powerOf10*，则 **MathLib.round** 将四舍五入到 *result* 中的小数位数。

integer 被定义为具有 INT 类型或者具有以下等效类型：BIN 类型，长度为 9 并且不带小数位。

示例： 在下面的示例中，项 *balance* 被四舍五入为最接近的千：

```
balance = 12345.6789;
rounder = 3;
balance = MathLib.round(balance, rounder);
// The value of balance is now 12000.0000
```

在下面的示例中，使用 *rounder* 值 -2 来将 *balance* 四舍五入为带有两个小数位：

```
balance = 12345.6789;
rounder = -2;
balance = mathLib.round(balance, rounder);
// The value of balance is now 12345.6800
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

sin()

系统函数 **MathLib.sin** 返回数字的正弦。结果位于 -1 到 1 的范围内。

```
MathLib.sin(numericField mathLibNumber in)
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。**MathLib.sin** 函数返回的值被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。在计算 *result* 之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.sin(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

sinh()

系统函数 **MathLib.sinh** 返回数字的双曲正弦。

```
MathLib.sinh(numericField mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。**MathLib.sinh** 函数返回的值被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。在计算 *result* 之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.sinh(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

sqrt()

算术函数 **MathLib.sqrt** 返回数字的平方根。此函数对任何大于等于零的数字执行运算。

```
MathLib.sqrt(numericField mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。**MathLib.sqrt** 函数返回的值被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。在计算 *result* 之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.sqrt(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

stringAsDecimal()

系统函数 **MathLib.stringAsDecimal** 接受字符值（如 "98.6"）并返回类型为 DECIMAL 的等效值。

```
MathLib.stringAsDecimal(numberAsText STRING in)  
returns (result DECIMAL)
```

result

类型为 DECIMAL 的值。接收字段可以具有任意小数位数和任意长度。

EGL 允许小数点的任一边最多有 32 位数字。小数点（如果有的话）将特定于 Java 语言环境。

有关将数字值指定给不同类型的字段的含义的详细信息，请参阅赋值。

numberAsText

字符字段或文字串，可包括初始符号字符。

示例:

```
myField = "-5.243";  
  
// result = -5.243  
result = MathLib.stringAsDecimal(myField);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 340 页的『赋值』

第 767 页的『EGL 库 MathLib』

stringAsFloat()

系统函数 **MathLib.stringAsFloat** 接受字符值（如 "98.6"）并返回类型为 FLOAT 的等效值。

```
MathLib.stringAsFloat(numberAsText STRING in)  
returns (result FLOAT)
```

result

类型为 FLOAT 的值。接收字段可以具有任意小数位数和任意长度。小数点（如果有的话）将特定于 Java 语言环境。

有关将数字值指定给不同类型的字段的含义的详细信息，请参阅赋值。

numberAsText

字符字段或文字串，可包括初始符号字符。

示例:

```
myField = "-5.243";  
  
// result = -5.243  
result = MathLib.stringAsFloat(myField);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 340 页的『赋值』

第 767 页的『EGL 库 MathLib』

stringAsInt()

系统函数 **MathLib.stringAsInt** 接受字符值（如 "98"）并返回类型为 BIGINT 的等效值。

```
MathLib.stringAsInt(numberAsText STRING in)  
returns (result BIGINT)
```

result

类型为 BIGINT 的值。

有关将数字值指定给不同类型的字段的含义的详细信息，请参阅赋值。

numberAsText

字符字段或文字串，可包括初始符号字符。

示例:

```
myField = "-5";  
  
// result = -5  
result = MathLib.stringAsInt(myField);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

tan()

系统函数 **MathLib.tan** 返回数字的正切。

```
MathLib.tan(numericField mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。**MathLib.tan** 函数返回的值被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。在计算 *result* 之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.tan(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

tanh()

系统函数 **MathLib.tanh** 返回数字的双曲正切。结果位于 -1 到 1 的范围内。

```
MathLib.tanh(numericField mathLibNumber in)  
returns (result mathLibTypeDependentResult)
```

result

任何数字或 HEX 项，如数学（系统字）中所述。**MathLib.tanh** 函数返回的值被转换为具有 *result* 的格式，并在 *result* 中返回。

numericField

任何数字或 HEX 项，如数学（系统字）中所述。在计算 *result* 之前，该项被转换为双精度浮点。

示例:

```
result = MathLib.tanh(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 767 页的『EGL 库 MathLib』

recordName.resourceAssociation

当程序对记录执行 I/O 操作时，该 I/O 是对一个物理文件进行的，该物理文件的名称包含在特定于记录的变量 **recordName.resourceAssociation** 中。该变量是根据在生成时使用的 **resourceAssociation** 部件进行初始化的；有关详细信息，请参阅**资源关联和文件类型**。在运行时，可以通过在 **resourceAssociation** 中指定另一个值来更改系统资源名称。

在大多数情况下，必须使用语法 **recordName.resourceAssociation**。但是，如果 EGL 可以确定您打算使用的记录，则不需要指定记录名，在下列每种情况下均如此：

- 在程序中，只对一条记录执行 I/O
- 在只对一条记录执行 I/O 的函数中使用 **resourceAssociation**
- 在程序中对多个记录执行 I/O，但所有记录都具有相同的文件名；在这种情况下，第一个作为 I/O 对象出现的记录被用作隐式限定符。

可以将 **resourceAssociation** 用作下列任何一项：

- 赋值语句的源或目标操作数
- **case**、**if** 或 **while** 语句中的逻辑表达式中的项
- **return** 或 **exit** 语句中的自变量

resourceAssociation 的特征如下所示：

基本类型

CHAR

数据长度

随文件类型的不同而有所变化

跨段保存?

是

定义注意事项

移到 *recordName*.**resourceAssociation** 中的值必须是系统和文件类型的有效系统资源名称，该系统和文件类型是在生成程序时指定的。如果多个记录指定了相同的文件名，则对任何具有该文件名的记录修改 **resourceAssociation** 时，都将更改程序中所有具有同一文件名的记录的 **resourceAssociation** 设置。

在修改该特定于记录的变量时，如果在 **resourceAssociation** 设置中标识的系统资源已打开，则在下列情况下将关闭该变量曾包含的系统资源：对一条记录运行 I/O 选项，该记录的 EGL 文件名与用于限定 **resourceAssociation** 的记录的 EGL 文件名相同。

如果两个程序正在使用同一个 EGL 文件名，则每个特定于记录的 **resourceAssociation** 变量都必须包含同一个值。否则，当打开新的系统资源时，将关闭先前打开的系统资源。

对于 **resourceAssociation** 与另一个值的比较，仅当完全匹配时才得到 true 结果。例如，如果用小写值来初始化 **resourceAssociation**，则小写值只与小写值相匹配。

在程序之间共享的文件： 可以在生成时或运行时设置系统资源名称：

在生成时

如果同一运行单元中的两个程序访问同一个逻辑文件，则在生成时必须对该逻辑文件指定相同的系统资源名称，以确保两个程序在运行时访问同一个物理文件。

在运行时

如果使用 *recordName*.**resourceAssociation**，则每个访问该文件的程序都必须为该文件设置 **resourceAssociation**。如果同一运行单元中的两个程序访问同一个逻辑文件，则每个程序都必须将 **resourceAssociation** 设置为相同的系统资源名称，以确保两个程序在运行时访问同一个物理文件。

如果系统资源由多个程序共享，则每个访问该资源的程序都必须设置 **resourceAssociation** 才能引用同一资源。并且，如果同一运行单元中的两个程序访问同一个逻辑文件，则每个程序在生成时都必须将 **resourceAssociation** 设置为同一系统资源名以确保两个程序在运行时访问同一系统资源。

MQ 记录： MQ 记录的系统资源名定义了队列管理器名和队列名。按以下格式指定名称：

queueManagerName:*queueName*

queueManagerName

队列管理器的名称。

queueName

队列的名称。

如上所示，名称是通过冒号隔开的。但是，可以省略 *queueManagerName* 和冒号。系统资源名称被用作特定于记录的 **resourceAssociation** 项的初始值，它标识与记录相关联的缺省队列。有关更多详细信息，请参阅 *MQSeries* 支持。

示例

```
if (process == 1)
  myrec.resourceAssociation = "myFile.txt";
else
  myrec.resourceAssociation = "myFile02.txt";
end
```

相关概念

第 242 页的『MQSeries 支持』

第 277 页的『资源关联和文件类型』

相关参考

EGL 库 ReportLib

ReportLib 是 EGL 报告库，这是一个系统库，用来建立将包含与 JasperReports 库交互所需的所有组件的框架。EGL 报告库包括下列组件：

- 用于下列用途的函数、变量和常量：
 - 与 JasperReports 库函数交互作用
 - 定义、设置和检索报告的数据源
 - 将填写的报告导出至不同文件格式
 - 处理报告的内容并处理报告数据
- 一些记录，它们包含存储报告设计、填写的报告及导出的报告的文件的名称。
- 报告

报告库包括下列函数：

系统函数 / 调用	描述
<code>addReportParameter(report, parameterString, parameterValue)</code>	使用指定数据源填写报告。
<code>fillReport(report, source)</code>	以指定格式导出填写的报告。
<code>exportReport(report, format)</code>	将值添加至报告的参数列表。
<code>resetReportParameters(report)</code>	除去用于特定报告的所有参数。

只能在报告处理程序中调用下列函数：

系统函数 / 调用	描述
<code>addReportData(rd, dataSetName)</code>	将带有指定名称的报告数据对象添加至当前报告处理程序。
<code>result = getReportData(dataSetName)</code>	检索带有指定名称的报告数据记录。返回的值为 <code>ReportData</code> 类型。
<code>result = getReportParameter(parameter)</code>	从要填写的报告返回指定参数的值。
<code>result = getFieldValue(fieldName)</code>	对当前处理的行返回指定字段值。返回的值为 <code>ANY</code> 类型。

系统函数 / 调用	描述
<code>result = getReportVariableValue(variable)</code>	从要填写的报告返回指定变量的值。返回的值为 ANY 类型。
<code>setReportVariableValue(variable, value)</code>	将指定变量的值设置为提供的值。

注: 如果删除 EGL 报告, 则必须除去对该报告的所有引用。

相关概念

第 191 页的『数据源』

第 190 页的『EGL 报告创建过程概述』

第 189 页的『EGL 报告概述』

addReportData()

系统函数 **ReportLib.addReportData** 将带有指定名称的报告数据对象添加至当前报告处理程序。

```
ReportLib.addReportData(
  rd ReportData in,
  dataSetName STRING in)
```

rd

dataSetName

相关概念

第 689 页的『EGL 函数的语法图』

第 189 页的『EGL 报告概述』

第 190 页的『EGL 报告创建过程概述』

相关参考

第 788 页的『EGL 库 ReportLib』

『addReportParameter()』

第 791 页的『fillReport()』

第 790 页的『exportReport()』

addReportParameter()

ReportLib.addReportParameter 函数的语法图如下所示:

```
ReportLib.addReportParameter(
  report Report in,
  parameterString STRING in,
  parameterValue any in)
```

report

报告的名称。

parameterString

要在报告中使用的参数。

parameterValue

在 *parameterString* 中指定的参数的值。

在填写报告之前，EGL 可传递一组参数，它们会确定要在报告中使用的值或覆盖在 XML 报告设计中指定的参数。**ReportLib.addReportParameter** 函数将指定参数的值添加至报告的参数列表。

注：有关 JasperReports 参数和数据类型的信息，请参阅 JasperReports 文档。

相关概念

第 689 页的『EGL 函数的语法图』

EGL 报告概述

EGL 报告创建过程概述

相关参考

EGL 报告库

ReportLib.fillReport 函数

ReportLib.exportReport 函数

ReportLib.resetReportParameters 函数

exportReport()

系统函数 **ReportLib.exportReport** 以指定格式导出填写的报告。

下图说明该函数的语法：

```
ReportLib.exportReport(  
    report Report in,  
    format ExportFormat in)
```

report

将导出的报告。

format

已导出报告的格式和文件扩展名。

下列值属于枚举 **ExportFormat**：

csv

输出显示用逗号隔开一个值与下一个值；**csv** 表示用逗号隔开的值。

html

输入为 HTML 格式。

pdf

输出为 Adobe Acrobat PDF 格式。

text

输入为 ASCII 文本格式。

相关概念

第 189 页的『EGL 报告概述』

第 190 页的『EGL 报告创建过程概述』

第 441 页的『EGL 中的枚举』

第 689 页的『EGL 函数的语法图』

相关任务

第 207 页的『导出报告』

相关参考

第 789 页的『addReportParameter()』

第 788 页的『EGL 库 ReportLib』

『fillReport()』

第 793 页的『resetReportParameters()』

fillReport()

ReportLib.fillReport 函数的语法图如下所示:

```
ReportLib.fillReport(  
  report Report in,  
  source DataSource in)
```

report

将使用数据填充该报告。

source

用于填充报告的数据的源。

考虑以下示例，它显示类型为 **reportData** 的变量如何与报告相关联:

```
eglReport    Report;  
eglReportData ReportData;  
eglReport.reportData = eglReportData;
```

source 指示要在类型为 **ReportData** 的变量中使用的字段。*source* 的每个值不是字段名，而是枚举 **DataSource** 中的值:

databaseConnection

使用 **reportData** 变量的 **connectionName** 字段中引用的变量，如以下示例中所示:

```
eglReportData.connectionName = "mycon";
```

在此情况下，访问数据的 SQL 语句位于报告设计文件中，该文件是在 EGL 外部创建的。

reportData

使用 **reportData** 变量的 **data** 字段中引用的变量，如以下示例中所示:

```
// an array of records, with data  
myRecords customerRecord[];  
  
eglReportData.data = myRecords;
```

sqlStatement

使用 **reportData** 变量的 **sqlStatement** 字段中标识的 SQL 语句，如以下示例中所示:

```
mySQLString = "Select * From MyTable";  
eglReportData.sqlStatement = mySQLString;
```

下面是一个示例调用:

```
ReportLib.fillReport (eglReport, DataSource.sqlStatement);
```

相关概念

第 189 页的『EGL 报告概述』

- 第 190 页的『EGL 报告创建过程概述』
- 第 441 页的『EGL 中的枚举』
- 第 689 页的『EGL 函数的语法图』

相关参考

- 第 191 页的『数据源』
- 第 788 页的『EGL 库 ReportLib』
- 第 789 页的『addReportParameter()』
- 第 790 页的『exportReport()』
- 第 793 页的『resetReportParameters()』

getFieldValue()

ReportLib.getFieldValue 函数对当前处理的行返回指定字段值。

```
ReportLib.getFieldValue(fieldName STRING in)  
returns (result ANY)
```

result

fieldName

相关概念

- 第 689 页的『EGL 函数的语法图』
- 第 189 页的『EGL 报告概述』
- 第 190 页的『EGL 报告创建过程概述』

相关参考

- 第 789 页的『addReportParameter()』
- 第 791 页的『fillReport()』
- 第 788 页的『EGL 库 ReportLib』
- 第 790 页的『exportReport()』

getReportData()

系统函数 **ReportLib.getReportData** 检索带有指定名称的报告数据记录。

```
ReportLib.getReportData(dataSetName STRING in)  
returns (result ReportData)
```

result

dataSetName

相关概念

- 第 689 页的『EGL 函数的语法图』
- 第 189 页的『EGL 报告概述』
- 第 190 页的『EGL 报告创建过程概述』

相关参考

- 第 789 页的『addReportParameter()』
- 第 788 页的『EGL 库 ReportLib』
- 第 790 页的『exportReport()』
- 第 791 页的『fillReport()』

getReportParameter()

ReportLib.getReportParameter 函数从要填写的报告返回指定参数的值。

```
ReportLib.getReportParameter(parameter STRING in)  
returns (result ANY)
```

result

parameter

相关概念

第 689 页的『EGL 函数的语法图』

第 189 页的『EGL 报告概述』

第 190 页的『EGL 报告创建过程概述』

相关参考

第 788 页的『EGL 库 ReportLib』

第 789 页的『addReportParameter()』

第 791 页的『fillReport()』

第 790 页的『exportReport()』

getReportVariableValue()

系统函数 **ReportLib.getReportVariableValue** 返回正在填写的报告中的指定变量的值。

```
ReportLib.getReportVariableValue(variable STRING in)  
returns (result ANY)
```

result

variable

相关概念

第 689 页的『EGL 函数的语法图』

第 189 页的『EGL 报告概述』

第 190 页的『EGL 报告创建过程概述』

相关参考

第 789 页的『addReportParameter()』

第 788 页的『EGL 库 ReportLib』

第 790 页的『exportReport()』

第 791 页的『fillReport()』

resetReportParameters()

ReportLib.resetReportParameters 函数的语法图如下所示:

```
ReportLib.resetReportParameters(report Report in)
```

report

包含想要除去的参数的报告的名称。

ReportLib.resetReportParameters 函数除去用于特定报告的所有 EGL 参数。

相关概念

- 第 689 页的『EGL 函数的语法图』
- 第 189 页的『EGL 报告概述』
- 第 190 页的『EGL 报告创建过程概述』

相关参考

- 第 788 页的『EGL 库 ReportLib』
- 第 789 页的『addReportParameter()』
- 第 790 页的『exportReport()』
- 第 791 页的『fillReport()』

setReportVariableValue()

ReportLib.setReportVariableValue 函数将指定变量的值设置为提供给该函数的值。

```
ReportLib.setReportVariableValue(  
    variable STRING in,  
    value Any in)
```

variable

value

相关概念

- 第 689 页的『EGL 函数的语法图』
- 第 190 页的『EGL 报告创建过程概述』
- 第 189 页的『EGL 报告概述』

相关参考

- 第 789 页的『addReportParameter()』
- 第 788 页的『EGL 库 ReportLib』
- 第 791 页的『fillReport()』
- 第 790 页的『exportReport()』

EGL 库 StrLib

下表显示库 **StrLib** 中的系统函数，并且该表后面的表将显示该库中的变量和常量。

系统函数和调用	描述
<i>result</i> = characterAsInt (<i>text</i>)	将字符串转换为与字符表达式中的第一个字符相对应的整数字符串。
<i>result</i> = clip (<i>text</i>)	删除返回的字符串的结束位置的结尾空格和空值
<i>result</i> = compareStr (<i>target</i> , <i>targetSubstringIndex</i> , <i>targetSubstringLength</i> , <i>source</i> , <i>sourceSubstringIndex</i> , <i>sourceSubstringLength</i>)	在运行时，根据两个子串的 ASCII 或 EBCDIC 顺序来对它们进行比较，并返回值 (-1、0 或 1) 以指出哪个值较大。
<i>result</i> = concatenate (<i>target</i> , <i>source</i>)	将 <i>target</i> 与 <i>source</i> 并置；将新字符串放在 <i>target</i> 中；并返回一个整数以指出 <i>target</i> 的长度是否足以包含新字符串
<i>result</i> = concatenateWithSeparator (<i>target</i> , <i>source</i> , <i>separator</i>)	通过在 <i>target</i> 与 <i>source</i> 之间插入 <i>separator</i> 来将它们并置；将新字符串放到 <i>target</i> 中；并返回一个整数以指出 <i>target</i> 的长度是否足以包含新字符串

系统函数和调用	描述
<code>copyStr (target, targetSubstringIndex, targetSubstringLength, source, sourceSubstringIndex, sourceSubstringLength)</code>	将一个子串复制至另一个子串
<code>result = findStr (source, sourceSubstringIndex, sourceSubstringLength, searchString)</code>	搜索子串在字符串中的第一次出现
<code>result = formatDate (dateValue [, dateFormat])</code>	定义日期值的格式并返回类型为 STRING 的值。缺省格式为当前语言环境中指定的格式。
<code>result = formatNumber (numericExpression, numericFormat)</code>	返回格式字符串形式的数字。
<code>result = formatTime (timeValue [, timeFormat])</code>	将参数的格式定义为时间值并返回类型为 STRING 的值。缺省格式为当前语言环境中指定的格式。
<code>result = formatTimeStamp (timeStampValue [, timeStampFormat])</code>	将参数的格式定义为时间戳记值并返回类型为 STRING 的值。DB2 格式为缺省格式。
<code>result = getNextToken (target, source, sourceSubstringIndex, sourceStringLength, characterDelimiter)</code>	在字符串中搜索下一个记号并将该记号复制至 <i>target</i>
<code>result = integerAsChar (integer)</code>	将整数字符串转换为字符串。
<code>result = lowerCase (text)</code>	将字符串中的所有大写值转换为小写值。数字和现有小写值不受影响。
<code>setBlankTerminator (target)</code>	将字符串中的 NULL 终止符以及任何后续字符都替换为空格，以使 C 或 C++ 程序返回的字符串值在 EGL 生成的程序中能够正确地工作
<code>setNullTerminator (target)</code>	将字符串中的所有结尾空格更改为 NULL
<code>setSubStr (target, targetSubstringIndex, targetSubstringLength, source)</code>	用指定的字符替换子串中的每个字符
<code>result =spaces (characterCount)</code>	返回指定长度的字符串。
<code>result = strlen (source)</code>	返回一个项中的字节数，不包括任何结尾空格或 NULL
<code>result = textLen (source)</code>	返回文本表达式中的字节数，不包括任何结尾空格或 NULL
<code>result = upperCase (characterItem)</code>	将字符串中的所有小写值转换为大写值。数字和现有大写值不受影响。

下表显示库 **StrLib** 中的系统变量。

系统变量	描述
<code>defaultDateFormat</code>	指定 defaultDateFormat 的值，它是一种掩码，可以用来创建由函数 StrLib.formatDate 返回的字符串。
<code>defaultMoneyFormat</code>	指定 defaultMoneyFormat 的值，它是一种掩码，可以用来创建由函数 StrLib.formatNumber 返回的字符串。

系统变量	描述
defaultNumericFormat	指定 defaultNumericFormat 的值，它是一种掩码，可以用来创建由函数 StrLib.formatNumber 返回的字符串。
defaultTimeFormat	指定 defaultTimeFormat 的值，它是一种掩码，可以用来创建由函数 StrLib.formatTime 返回的字符串。
defaultTimestampFormat	指定 defaultTimestampFormat 的值，它是一种掩码，可以用来创建由函数 StrLib.formatTimestamp 返回的字符串。

下表显示库 **StrLib** 中的系统常量。所有常量都属于 STRING 类型。

系统变量	描述
db2TimestampFormat	模式 <i>yyyy-MM-dd-HH.mm.ss.ffffff</i> ，这是 IBM DB2 缺省时间戳记格式。
eurDateFormat	模式 <i>dd.MM.yyyy</i> ，这是 IBM 欧洲标准日期格式。
eurTimeFormat	模式 <i>HH.mm.ss</i> ，这是 IBM 欧洲标准时间格式。
isoDateFormat	模式 <i>yyyy-MM-dd</i> ，这是国际标准化组织（ISO）指定的日期格式。
isoTimeFormat	模式 <i>HH.mm.ss</i> ，这是国际标准化组织（ISO）指定的时间格式。
jisDateFormat	模式 <i>yyyy-MM-dd</i> ，这是日本工业标准日期格式。
jisTimeFormat	模式 <i>HH:mm:ss</i> ，这是日本工业标准时间格式。
odbcTimestampFormat	模式 <i>yyyy-MM-dd HH:mm:ss.ffffff</i> ，这是 ODBC 时间戳记格式。
usaDateFormat	模式 <i>MM/dd/yyyy</i> ，这是 IBM 美国标准日期格式。
usaTimeFormat	模式 <i>hh:mm AM</i> ，这是 IBM 美国标准时间格式。

相关参考

第 804 页的『formatDate()』

第 805 页的『formatNumber()』

第 805 页的『formatTime()』

第 806 页的『formatTimeStamp()』

characterAsInt()

字符串格式的函数 **StrLib.characterAsInt** 将字符串转换为对应字符表达式中的第一个字符的整数字符串。

```
StrLib.characterAsInt(text STRING in)
returns (result INT)
```

result

类型为 INT 的变量。

text

返回类型为 CHAR 的字符串的文字、变量或表达式。

要将整数字符串转换为字符串，使用 **StrLib.integerAsChar** 字符串格式函数。

相关参考

第 794 页的『EGL 库 StrLib』

第 809 页的『integerAsChar()』

clip()

字符串格式函数 **StrLib.clip** 删除返回的字符串的结束位置的结尾空格和空值。

```
StrLib.clip(text STRING in)  
returns (result STRING)
```

result

一个字符串。

text

返回类型为 CHAR 的字符串的文字、变量或表达式。

相关参考

第 794 页的『EGL 库 StrLib』

compareStr()

系统函数 **StrLib.compareStr** 在运行时根据两个子串的 ASCII 或 EBCDIC 顺序来对它们作比较。

```
StrLib.compareStr(  
  target VagText in,  
  targetSubStringIndex INT in,  
  targetSubStringLength INT in,  
  source VagText in,  
  sourceSubStringIndex INT in,  
  sourceSubStringLength INT in )  
returns (result INT)
```

result

一个数字项，它接收函数返回的下列其中一个值（该值被定义为具有 INT 类型或者具有以下等效类型：BIN 类型，长度为 9 并且不带小数位）：

- 1** 基于 *target* 的子串小于基于 *source* 的子串
- 0** 基于 *target* 的子串等于基于 *source* 的子串
- 1** 基于 *target* 的子串大于基于 *source* 的子串

target

从中派生目标子串的字符串。可以是项或文字。

targetSubStringIndex

在假定 *target* 中的第一个字节具有下标值 1 的情况下，标识子串在 *target* 中的起始字节。此下标可以是整数文字。另外，此下标可以是被定义为具有 INT 类型或者具有以下等效类型的项：BIN 类型，长度为 9 并且不带小数位。

targetSubStringLength

标识从 *target* 派生的子串中的字节数。长度可以是整数文字。另外，此下标可以是被定义为具有 INT 类型或者具有以下等效类型的项: BIN 类型, 长度为 9 并且不带小数位。

source

从中派生源子串的字符串。可以是项或文字。

sourceSubStringIndex

在假定 *source* 中的第一个字节具有下标值 1 的情况下, 标识子串在 *source* 中的起始字节。此下标可以是整数文字。另外, 此下标可以是被定义为具有 INT 类型或者具有以下等效类型的项: BIN 类型, 长度为 9 并且不带小数位。

sourceSubStringLength

标识从 *source* 派生的子串中的字节数。长度可以是整数文字。另外, 此下标可以是被定义为具有 INT 类型或者具有以下等效类型的项: BIN 类型, 长度为 9 并且不带小数位。

执行子串值的逐字节二进制比较。如果子串的长度不同, 则在进行比较之前, 用空格对较短的子串进行填充。

定义注意事项: 在 **sysVar.errorCode** 中返回下列值:

- 8** 下标小于 1 或大于字符串长度。
- 12** 长度小于 1。
- 20** 双字节下标无效。DBCHAR 或 UNICODE 字符串的下标指向双字节字符的中间
- 24** 双字节长度无效。DBCHAR 或 UNICODE 字符串的以字节计的长度是奇数 (双字节长度必须总是偶数)。

示例:

```
target = "123456";
source = "34";
result =
  StrLib.compareStr(target,3,2,source,1,2);
// result = 0
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 794 页的『EGL 库 StrLib』

concatenate()

系统函数 **StrLib.concatenate** 将两个字符串并置。

```
StrLib.concatenate(
  target VagText inOut,
  source VagText in)
returns (result INT)
```

result

一个数字项, 它接收函数返回的下列其中一个值 (该值被定义为具有 INT 类型或者具有以下等效类型: BIN 类型, 长度为 9 并且不带小数位):

- 1 并置的字符串太长，在目标项中放不下，该字符串已被截断，如后文所述
- 0 并置的字符串在目标项中放得下

target

目标项

source

源项或文字

当并置两个字符串时，将发生下列情况：

1. 将从目标字符串中删除任何结尾空格或 NULL。
2. 将对步骤 1 生成的字符串追加源字符串。
3. 如果步骤 2 生成的字符串比目标字符串项长，则会将其截断。如果它比目标项短，则会被填充空格。

示例：

```
phrase = "and/ "; // CHAR(7)
or      = "or";
result =
  StrLib.concatenate(phrase,or);
if (result == 0)
  print phrase; // phrase = "and/or "
end
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 794 页的『EGL 库 StrLib』

concatenateWithSeparator()

系统函数 **StrLib.concatenateWithSeparator** 将两个字符串并置，并在它们之间插入分隔符字符串。如果目标字符串的初始长度为零（不计结尾空格和 NULL），则省略分隔符，并将源字符串复制至目标字符串。

```
StrLib.concatenateWithSeparator(
  target VagText inOut,
  source VagText in,
  separator VagText in)
returns (result INT)
```

result

一个数字项，它接收函数返回的下列其中一个值（该值被定义为具有 INT 类型或者具有以下等效类型：BIN 类型，长度为 9 并且不带小数位）：

- 0 并置的字符串在目标项中放得下。
- 1 并置的字符串太长，在目标项中放不下，该字符串已被截断，如后文所述

target

目标项。

source

源项或文字。

separator

分隔符项或文字。

将从 *target* 中截断结尾空格和 NULL; 然后, 对截断的值追加 *separator* 字符串和 *source*。如果并置比目标所允许的长度要长, 则发生截断。如果并置比目标所允许的长度要短, 则用空格对并置的值进行填充。

示例:

```
phrase = "and"; // CHAR(7)
or     = "or";
result =
  StrLib.concatenateWithSeparator(phrase,or,"/");
if (result == 0)
  print phrase; // phrase = "and/or "
end
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 794 页的『EGL 库 StrLib』

copyStr()

系统函数 **StrLib.copyStr** 将一个子串复制至另一个子串。

```
StrLib.copyStr(
  target VagText inOut,
  targetSubstringIndex INT in,
  targetSubstringLength INT in,
  source VagText in,
  sourceSubstringIndex INT in,
  sourceSubstringLength INT in)
```

target

从中派生目标子串的字符串。可以是项或文字。

targetSubstringIndex

在假定 *target* 中的第一个字节具有值 1 的情况下, 标识 *target* 中的起始字节。此下标可以是整数文字。另外, 此下标可以是被定义为具有 INT 类型或者具有以下等效类型的项: BIN 类型, 长度为 9 并且不带小数位。

targetSubstringLength

标识从 *target* 派生的子串中的字节数。长度可以是整数文字。另外, 长度可以是被定义为具有 INT 类型或者具有以下等效类型的项: BIN 类型, 长度为 9 并且不带小数位。

source

从中派生源子串的字符串。可以是项或文字。

sourceSubstringIndex

在假定 *source* 中的第一个字节具有值 1 的情况下, 标识子串在 *source* 中的起始字节。此下标可以是整数文字。另外, 此下标可以是被定义为具有 INT 类型或者具有以下等效类型的项: BIN 类型, 长度为 9 并且不带小数位。

sourceSubstringLength

标识从 *source* 派生的子串中的字节数。长度可以是整数文字。另外, 长度可以是被定义为具有 INT 类型或者具有以下等效类型的项: BIN 类型, 长度为 9 并且不带小数位。

如果源比目标长, 则将源截断。如果源比目标短, 则在右边用空格对源值进行填充。

定义注意事项: 在 `sysVar.errorCode` 中返回下列值:

- 8** 下标小于 1 或大于字符串长度。
- 12** 长度小于 1。
- 20** 双字节下标无效。DBCHAR 或 UNICODE 字符串的下标指向双字节字符的中间。
- 24** 双字节长度无效。DBCS 或 UNICODE 字符串的以字节计的长度是奇数（双字节长度必须总是偶数）。

示例:

```
target = "120056";
source = "34";
StrLib.copyStr(target,3,2,source,1,2);
// target = "123456"
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 794 页的『EGL 库 StrLib』

defaultDateFormat

系统变量 `StrLib.defaultDateFormat` 指定 `defaultDateFormat` 的值，它是一种掩码，可以用来创建由函数 `StrLib.formatDate` 返回的字符串。

`StrLib.defaultDateFormat` 的初始值就是 Java 运行时属性 `vgj.default.dateFormat` 的值。如果未设置该属性，则 `StrLib.defaultDateFormat` 的初始值为 `MM/dd/yyyy`。

有关时间掩码的特征的详细信息，请参阅 [日期、时间和时间戳记说明符](#)。

类型: STRING

相关参考

第 42 页的『日期、时间和时间戳记格式说明符』

第 794 页的『EGL 库 StrLib』

第 804 页的『formatDate()』

第 493 页的『Java 运行时属性（详细信息）』

defaultMoneyFormat

系统变量 `StrLib.defaultMoneyFormat` 指定 `defaultMoneyFormat` 的值，它是一种掩码，可以用来创建由函数 `StrLib.formatNumber` 返回的字符串。

`StrLib.defaultMoneyFormat` 的初始值就是 Java 运行时属性 `vgj.default.moneyFormat` 的值。如果未设置该属性，则 `StrLib.defaultMoneyFormat` 的初始值是空字符串。

有关数字掩码的特征的详细信息，请参阅 [formatNumber\(\)](#)。

类型: STRING

相关参考

第 794 页的『EGL 库 StrLib』

第 805 页的『formatNumber()』

第 493 页的『Java 运行时属性（详细信息）』

defaultNumericFormat

系统变量 **StrLib.defaultNumericFormat** 指定 **defaultNumericFormat** 的值，它是一种掩码，可以用来创建由函数 **StrLib.formatNumber** 返回的字符串。

StrLib.defaultNumericFormat 的初始值就是 Java 运行时属性 **vgj.default.numericFormat** 的值。如果未设置该属性，则 **StrLib.defaultNumericFormat** 的初始值是空字符串。

有关数字掩码的特征的详细信息，请参阅 *formatNumber()*。

类型: STRING

相关参考

第 794 页的『EGL 库 StrLib』

第 805 页的『formatNumber()』

第 493 页的『Java 运行时属性（详细信息）』

defaultTimeFormat

系统变量 **StrLib.defaultTimeFormat** 指定 **defaultTimeFormat** 的值，它是一种掩码，可以用来创建由函数 **StrLib.formatTime** 返回的字符串。不会在任何其它上下文中使用该变量。

StrLib.defaultTimeFormat 的初始值就是 Java 运行时属性 **vgj.default.timeFormat** 的值。如果未设置该属性，则 **StrLib.defaultTimeFormat** 的初始值为 *HH:mm:ss*。

有关时间掩码的特征的详细信息，请参阅 *日期、时间和时间戳记说明符*。

类型: STRING

相关参考

第 42 页的『日期、时间和时间戳记格式说明符』

第 794 页的『EGL 库 StrLib』

第 805 页的『formatTime()』

第 493 页的『Java 运行时属性（详细信息）』

defaultTimestampFormat

系统变量 **StrLib.defaultTimestampFormat** 指定 **defaultTimestampFormat** 的值，它是一种掩码，可以用来创建由函数 **StrLib.formatTimestamp** 返回的字符串。

StrLib.defaultTimestampFormat 的初始值就是 Java 运行时属性 **vgj.default.timestampFormat** 的值。如果未设置该属性，则 **StrLib.defaultTimestampFormat** 的初始值是空字符串。

有关时间戳记掩码的特征的详细信息，请参阅 *日期、时间和时间戳记说明符*。

类型: STRING

相关参考

第 42 页的『日期、时间和时间戳记格式说明符』

第 794 页的『EGL 库 StrLib』

第 806 页的『formatTimeStamp()』

第 493 页的『Java 运行时属性 (详细信息)』

findStr()

系统函数 **StrLib.findStr** 在字符串中搜索子串的第一次出现。

```
StrLib.findStr(  
    source VagText in,  
    sourceSubstringIndex INT inOut,  
    sourceSubStringLength INT in,  
    searchString VagText in)  
returns (result INT)
```

result

一个数字项, 它接收函数返回的下列其中一个值 (该值被定义为具有 INT 类型或者具有以下等效类型: BIN 类型, 长度为 9 并且不带小数位):

- 1 找不到搜索字符串
- 0 找到搜索字符串

source

从中派生源子串的字符串。可以是项或文字。

sourceSubstringIndex

在假定 *source* 中的第一个字节具有下标值 1 的情况下, 标识子串在 *source* 中的起始字节。此下标可以是被定义为具有 INT 类型或者具有以下等效类型的项: BIN 类型, 长度为 9 并且不带小数位。

sourceStringLength

标识从 *source* 派生的子串中的字节数。此下标可以是被定义为具有 INT 类型或者具有以下等效类型的项: BIN 类型, 长度为 9 并且不带小数位。

searchString

要在源子串中搜索的字符串项或文字。在开始搜索之前, 截断搜索字符串中的结尾空格或空。

如果在源子串中找到 *searchString*, 则设置 *sourceSubstringIndex* 以指示其位置 (匹配子串在源子串中的起始字节)。否则, 不更改 *sourceSubstringIndex*。

定义注意事项: 在 *sysVar.errorCode* 中返回下列值:

- 8** 下标小于 1 或大于字符串长度。
- 12** 长度小于 1。
- 20** 双字节下标无效。DBCHAR 或 UNICODE 字符串的下标指向双字节字符的中间。
- 24** 双字节长度无效。DBCHAR 或 UNICODE 字符串的以字节计的长度是奇数 (双字节长度必须总是偶数)。

示例:

```

source = "123456";
sourceIndex = 1
sourceLength = 6
search = "34";
result =
  StrLib.findStr(source,sourceIndex,sourceLength,"34");
// result = 0, sourceIndex = 3

```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 794 页的『EGL 库 StrLib』

formatDate()

系统函数 **StrLib.formatDate** 会定义日期值的格式并返回类型为 **STRING** 的值。

```

StrLib.formatDate(
  dateValue DATE in
  [, dateFormat STRING in])
returns (result STRING)

```

result

类型为 **STRING** 的变量。

dateValue

要定义格式的值。可以是解析为日期值的任何表达式；例如，系统变量 **VGVar.currentGregorianDate**。

dateFormat

标识日期格式，如日期、时间和时间戳记说明符中所述。如果未对 *dateFormat* 指定任何值，EGL 运行时将使用 Java 语言环境中的日期格式。

可使用字符串、系统变量 **StrLib.defaultDateFormat**（如 *defaultDateFormat* 中所述）或下列任何常量：

StrLib.eurDateFormat

模式 *dd.MM.yyyy*，这是 IBM 欧洲标准日期格式

StrLib.isoDateFormat

模式 *yyyy-MM-dd*，这是国际标准化组织（ISO）指定的日期格式

StrLib.jisDateFormat

模式 *yyyy-MM-dd*，这是日本工业标准日期格式

StrLib.usaDateFormat

模式 *MM/dd/yyyy*，这是 IBM 美国标准日期格式

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 38 页的『DATE』

第 42 页的『日期、时间和时间戳记格式说明符』

第 801 页的『defaultDateFormat』

第 794 页的『EGL 库 StrLib』

formatNumber()

字符串函数 **StrLib.formatNumber** 返回格式字符串形式的数字。

```
StrLib.formatNumber(  
    numericExpression anyNumericExpression in,  
    numericFormat STRING in)  
returns (result STRING)
```

result

类型为 STRING 的变量。

numericExpression

要定义格式的数字值。可以是解析为数字的任何表达式。

numericFormat

定义如何格式化该数字的字符串。下表提供详细信息。该字符串是必需的，但您可以使用系统变量 **StrLib.defaultMoneyFormat** 或 **StrLib.defaultNumericFormat**。有关这些变量的详细信息，请参阅 *defaultMoneyFormat* 和 *defaultNumericFormat*。

有效字符如下所示：

- # 表示一位数字的占位符。
- * 将星号 (*) 用作前导零的填充字符。
- & 将零用作前导零的填充字符。
- # 将空格用作前导零的填充字符。
- < 使数字向左对齐。
- , 除非该位置包含前导零，否则使用根据语言环境确定的数字分隔符。
- . 使用根据语言环境确定的小数点。
- 使用减号 (-) 来表示小于 0 的值；使用空格来表示大于或等于 0 的值。
- + 使用减号 (-) 来表示小于 0 的值；使用加号 (+) 来表示大于或等于 0 的值。
- (在记帐时，适当地在负值前加上左圆括号。
-) 在记帐时，适当地在负值后加上右圆括号。
- \$ 根据语言环境在该值前加上适当的货币符号。
- @ 根据语言环境在该值后加上适当的货币符号。

相关参考

第 801 页的『defaultMoneyFormat』

第 802 页的『defaultNumericFormat』

第 794 页的『EGL 库 StrLib』

formatTime()

日期时间函数 **StrLib.formatTime** 定义时间值的格式并返回类型为 STRING 的值。

```
StrLib.formatTime(  
    aTime Time in  
    [, timeFormat STRING in  
    ])  
returns (result STRING)
```

result

类型为 `STRING` 的变量。

aTime

要定义格式的值。可以是解析为时间值的任何表达式；例如，系统变量 `DateTimeLib.currentTime`。

timeFormat

标识时间格式，如日期、时间和时间戳记说明符中所述。如果未对 *timeFormat* 指定任何值，EGL 运行时将使用 Java 语言环境中的时间格式。

可使用字符串、系统变量 `StrLib.defaultTimeFormat`（如 *defaultTimeFormat* 中所述）或下列任何常量：

eurTimeFormat

模式 *HH.mm.ss*，这是 IBM 欧洲标准时间格式。

isoTimeFormat

模式 *HH.mm.ss*，这是国际标准化组织（ISO）指定的时间格式。

jisTimeFormat

模式 *HH:mm:ss*，这是日本工业标准时间格式。

usaTimeFormat

模式 *hh:mm AM*，这是 IBM 美国标准时间格式。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 42 页的『日期、时间和时间戳记格式说明符』

第 802 页的『defaultTimeFormat』

第 794 页的『EGL 库 StrLib』

第 40 页的『TIME』

formatTimeStamp()

日期时间格式函数 `StrLib.formatTimeStamp` 将参数的格式定义为时间戳记值并返回类型为 `STRING` 的值。

```
StrLib.formatTimeStamp(  
    aTimeStamp TimeStamp in  
    [, timeStampFormat STRING in  
    ])  
returns (result STRING)
```

result

类型为 `STRING` 的变量。

aTimeStamp

要定义格式的 `TIMESTAMP` 值。可以是解析为 `TIMESTAMP` 值的任何表达式；例如，系统变量 `DateTimeLib.currentTimeStamp`。

timeStampFormat

标识日期格式，如日期、时间和时间戳记说明符中所述。

可使用字符串、系统变量 `StrLib.defaultTimestampFormat`（如 *defaultTimestampFormat* 中所述）或下列其中一个常量：

db2TimeStampFormat

模式 `yyyy-MM-dd-HH.mm.ss.ffffff`, 这是 IBM DB2 缺省时间戳记格式。

odbcTimeStampFormat

模式 `yyyy-MM-dd HH:mm:ss.ffffff`, 这是 ODBC 时间戳记格式。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 725 页的『currentTimeStamp()』

第 42 页的『日期、时间和时间戳记格式说明符』

第 802 页的『defaultTimeStampFormat』

第 794 页的『EGL 库 StrLib』

第 40 页的『TIMESTAMP』

getNextToken()

系统函数 **StrLib.getNextToken** 在子串中搜索记号并将该记号复制至目标项。

标记是用定界字符分隔的字符串。例如, 如果将空格 (“ ”) 和逗号 (“;”) 字符定义为定界符, 则可以将字符串 "CALL PROGRAM ARG1,ARG2,ARG3" 分为 5 个记号 "CALL"、"PROGRAM"、"ARG1"、"ARG2" 和 "ARG3"。

```
StrLib.getNextToken(  
    target VagText inOut,  
    source VagText in,  
    sourceSubstringIndex INT inOut,  
    sourceSubstringLength INT inOut,  
    characterDelimiter VagText in)  
returns (result INT)
```

result

被定义为具有 INT 类型或者具有以下等效类型的项: BIN 类型, 长度为 9 并且不带小数位。值是下列其中一项:

- +n 标记中的字符数。将把该记号从正在查看的子串复制至目标项。
- 0 在正在查看的子串中没有记号。
- 1 记号在被复制至目标项时被截断。

target

CHAR、DBCHAR、HEX、MBCHAR 或 UNICODE 类型的目标项。

source

CHAR、DBCHAR、HEX、MBCHAR 或 UNICODE 类型的源项。可以是那些除 UNICODE 之外的任何类型的文字。

sourceSubstringIndex

在假定 *source* 中的第一个字节具有值 1 的情况下, 标识作为定界符搜索开始位置的起始字节。 *sourceSubstringIndex* 可以是被定义为具有 INT 类型或者具有以下等效类型的项: BIN 类型, 长度为 9 并且不带小数位。如果找到记号, 则将 *sourceSubstringIndex* 中的值更改为记号后的第一个字符的下标。

sourceSubstringLength

指示所查看的子串中的字节数。 *sourceSubstringLength* 可以是被定义为具有 INT 类

型或者具有以下等效类型的项: BIN 类型, 长度为 9 并且不带小数位。如果找到记号, 则 *sourceSubstringLength* 中的值更改为在返回的记号之后开始的子串中的字节数。

characterDelimiter

一个或多个定界字符, 在各个定界字符之间没有分隔字符。可以是 CHAR、DBCHAR、HEX、MBCHAR 或 UNICODE 类型的项。可以是那些除 UNICODE 之外的任何类型的文字。

可以调用一连串的调用以检索子串中的每个记号, 而不必重置 *sourceSubstringIndex* 和 *sourceSubstringLength* 的值, 如后面的示例所示。

错误状态: 在 **SysVar.errorCode** 中返回下列值:

- 8** *sourceSubstringIndex* 小于 1 或大于所查看的子串中的字节数。
- 12** *sourceSubstringLength* 小于 1。
- 20** 对于 DBCHAR 或 UNICODE 字符串, *sourceSubstringIndex* 中的值指的是双字节字符中的中间。
- 24** DBCHAR 或 UNICODE 字符串的 *sourceSubstringLength* 值是奇数 (双字节长度必须总是偶数)。

示例:

```
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
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 794 页的『EGL 库 StrLib』

integerAsChar()

字符串格式函数 **StrLib.integerAsChar** 将整数字符串转换为字符串。

```
StrLib.integerAsChar(integer INT in)  
returns (result STRING)
```

result

类型为 STRING 的变量。

integer

返回类型为 BIGINT、INT 或 SMALLINT 的整数的文字、变量或表达式。

要将字符串转换为整数字符串，使用 **StrLib.characterAsInt** 字符串格式函数。

相关参考

第 794 页的『EGL 库 StrLib』

第 796 页的『characterAsInt()』

lowerCase()

字符串格式函数 **StrLib.lowerCase** 将字符串中的所有大写值转换为小写值。数字和现有小写值不受影响。

```
StrLib.lowerCase(text STRING in)  
returns (result STRING)
```

result

类型为 STRING 的变量。

text

返回类型为 CHAR 的字符串的文字、变量或表达式。

StrLib.lowerCase 函数对类型为 DBCHAR 或 MBCHAR 的项中的双字节字符不起作用。

要将小写值转换为大写值，使用 **StrLib.upperCase** 字符串格式函数。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 794 页的『EGL 库 StrLib』

第 812 页的『upperCase()』

setBlankTerminator()

系统函数 **StrLib.setBlankTerminator** 将 NULL 终止符和任何后续字符更改为空格。**StrLib.setBlankTerminator** 将 C 或 C++ 程序返回的字符串值更改为能够在 EGL 程序中正确工作的字符值。

```
StrLib.setBlankTerminator(target VagText inOut)
```

target

目标字符串项。如果在 *targetString* 中找不到 NULL，则此函数没有任何效果。

示例:

```
StrLib.setBlankTerminator(target);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 794 页的『EGL 库 StrLib』

setNullTerminator()

系统函数 **StrLib.setNullTerminator** 将字符串中的所有结尾空格更改为 NULL。在将一个项传递至期望以 null 结束的字符串作为自变量的 C 或 C++ 程序时，可以使用 **StrLib.setNullTerminator** 来转换该项。

```
StrLib.setNullTerminator(target VagText inOut)
```

target

要转换的字符串

将在目标字符串中搜索结尾空格和 NULL。找到的任何空格都将被更改为 NULL。

定义注意事项: 可以在 **sysVar.errorCode** 中返回下列值:

16 字符串的最后一个字节不是空格或 NULL

示例:

```
StrLib.setNullTerminator(myItem01);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 794 页的『EGL 库 StrLib』

setSubStr()

系统函数 **StrLib.setSubStr** 用指定的字符替换子串中的每个字符。

```
StrLib.setSubStr(  
  target VagText inOut,  
  targetSubstringIndex INT in,  
  targetSubstringLength INT in,  
  source)
```

target

更改的项。

targetSubstringIndex

在假定 *target* 中的第一个字节具有下标值 1 的情况下，标识子串在 *target* 中的起始字节。此下标可以是整数文字。另外，此下标可以是被定义为具有 INT 类型或者具有以下等效类型的项：BIN 类型，长度为 9 并且不带小数位。

targetSubstringLength

标识从 *target* 派生的子串中的字节数。长度可以是整数文字。另外，长度可以是被定义为具有 INT 类型或者具有以下等效类型的项：BIN 类型，长度为 9 并且不带小数位。

source

如果目标项是 CHAR、MBCHAR 或 HEX，则源项必须是单字节 CHAR、MBCHAR 或 HEX 项或者 CHAR 文字。如果目标是 DBCHAR 或 UNICODE 项，则源必须是单字符 DBCHAR 或 UNICODE 项。

定义注意事项： 在 SysVar.errorCode 中返回下列值：

- 8** 下标小于 1 或大于字符串长度
- 12** 长度小于 1
- 20** 双字节下标无效。DBCHAR 或 UNICODE 字符串的下标指向双字节字符的中间
- 24** 双字节长度无效。DBCHAR 或 UNICODE 字符串的以字节计的长度是奇数（双字节长度必须总是偶数）

示例：

```
StrLib.setSubStr(target,12,5," ");
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 794 页的『EGL 库 StrLib』

spaces()

系统函数 **StrLib.spaces** 返回由指定数目的空格组成的字符串。

```
StrLib.spaces(characterCount INT in)  
returns (result STRING)
```

result

类型为 STRING 的变量。

characterCount

要返回的空格字符串的长度。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 794 页的『EGL 库 StrLib』

strlen()

系统函数 **StrLib strlen** 返回一个项中的字节数，不包括任何结尾空格或 NULL。

```
StrLib strlen(source VagText in)
returns (result INT)
```

result

被定义为具有 INT 类型或者具有以下等效类型的项: BIN 类型，长度为 9 并且不带小数位。

source

要度量的字符串项或文字。

示例:

```
length = StrLib strlen(source);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 794 页的『EGL 库 StrLib』

textLen()

系统函数 **StrLib textLen** 返回文本表达式中的字符数，不包括任何结尾空格或 NULL。

```
StrLib textLen(source STRING in)
returns (result INT)
```

result

被定义为具有 INT 类型或者具有以下等效类型的项: BIN 类型，长度为 9 并且不带小数位。

source

有意义的文本表达式。

示例:

```
length = StrLib textLen(source);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 794 页的『EGL 库 StrLib』

第 462 页的『文本表达式』

upperCase()

字符串格式函数 **StrLib upperCase** 将字符串中的所有小写值转换为大写值。数字和现有大写值不受影响。

```
StrLib upperCase(text STRING in)
returns (result STRING)
```

result

类型为 STRING 的变量。

text

返回类型为 CHAR 的字符串的文字、变量或表达式。

StrLib.upperCase 函数对类型为 DBCHAR 或 MBCHAR 的项中的双字节字符不起作用。

要将字符串转换为小写，使用 **StrLib.lowerCase** 字符串格式函数。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 794 页的『EGL 库 StrLib』

第 809 页的『lowerCase()』

EGL 库 SysLib

功能	描述
<code>beginDatabaseTransaction([database])</code>	开始关系数据库事务，但仅当 EGL 运行时不会自动落实更改时才会如此。
<code>result = bytes(field)</code>	返回指定内存区的字节数。
<code>calculateChkDigitMod10 (text, checkLength, result)</code>	将模 10 校验数位放入以一系列整数开始的字符项中。
<code>calculateChkDigitMod11 (text, checkLength, result)</code>	将模 11 校验数位放入以一系列整数开始的字符项中。
<code>callCmd (commandString[, modeString])</code>	运行系统命令并等待命令完成。
<code>commit()</code>	保存自上次落实后对数据库、MQSeries 消息队列和 CICS 可恢复文件所作的更新。生成的 Java 程序或包装器还保存由基于 CICS 的远程 COBOL 程序所作的更新（包括对 CICS 可恢复文件所作的更新），但仅当对远程 COBOL 程序进行的调用涉及客户机控制的工作单元时才这样做，如 <code>callLink</code> 元素中的 <code>luwControl</code> 所述。
<code>result = conditionAsInt (booleanExpression)</code>	接受逻辑表达式（如 <code>myVar == 6</code> ），如果表达式求值为 true，则返回 1，如果表达式求值为 0，则返回 0。
<code>connect (database, userID, password[, commitScope[, disconnectOption[, isolationLevel[, commitControl]]]])</code>	关闭所有游标，释放锁定，结束所有现有连接并连接至数据库。
<code>convert (target, direction, conversionTable)</code>	在 EBCDIC（主机）与 ASCII（工作站）格式之间转换数据，或者在一种格式中执行代码页转换。
<code>defineDatabaseAlias (alias, database)</code>	创建一个别名，该别名可用于来建立与代码已连接至的数据库的新连接。
<code>disconnect ([database])</code>	与指定的数据库断开连接，或者（如果未指定数据库的话）与当前数据库断开连接。
<code>disconnectAll ()</code>	与当前连接的所有数据库断开连接。

功能	描述
<code>errorLog ()</code>	将文本复制到由系统函数 SysLib.startLog 启动的错误日志中。
<code>result = getCmdLineArg (index)</code>	从 EGL 程序涉及的自变量列表中返回指定自变量。指定自变量返回字符串值形式的自变量。
<code>result = getCmdLineArgCount ()</code>	返回用于启动 EGL 主程序的自变量的数目。
<code>result = getMessage (key [, insertArray])</code>	从 Java 运行时属性 <code>vgj.message.file</code> 中引用的文件中返回一条消息。
<code>result = getProperty(propertyName)</code>	检索 Java 运行时属性的值。如果找不到指定属性，则函数返回空字符串 ("")。
<code>loadTable (filename, insertintoClause[, delimiter])</code>	将文件中的数据装入到关系数据库中。
<code>result = maximumSize (arrayName)</code>	返回数据项或记录的动态数组中可以包含的最大行数；明确地说，此函数返回数组属性 maxSize 的值。
<code>queryCurrentDatabase (product, release)</code>	返回当前连接的数据库的产品号和发行版号。
<code>rollback ()</code>	撤销自上次落后对数据库和 MQSeries 消息队列所作的更新。该撤销发生在任何 EGL 生成的应用程序中。
<code>setCurrentDatabase (database)</code>	使指定的数据库成为当前活动数据库。
<code>setError (itemInError, msgKey[, itemInsert])</code> <code>setError (this, msgKey[, itemInsert])</code> <code>setError (msgText)</code>	将消息与 PageHandler 或用户界面记录中的某项相关联，或者与 PageHandler 或用户界面记录整体相关联。该消息被放在 JSP 中的 JSF 消息或消息标记位置，并且是在相关 Web 页面显示时显示的。
<code>setLocale (languageCode, countryCode[, variant])</code>	用于 PageHandler 以及在 Web 应用程序中运行的程序。
<code>setRemoteUser (userID, passWord)</code>	设置从 Java 程序调用远程程序时使用的用户标识和密码。
<code>result = size (arrayName)</code>	返回指定的数据表中的行数或指定的数组中的元素数。该数组可以是结构项数组、数据项或记录的静态数组或者数据项或记录的动态数组。
<code>startCmd (commandString[, modeString])</code>	运行系统命令并且不等待命令完成。
<code>startLog (logFile)</code>	打开错误日志。每当程序调用 SysLib.errorLog 时，就会将文本写入到该日志中。
<code>startTransaction (termID[, prID[, termID]])</code>	以异步方式调用主程序，将该程序与打印机或终端设备相关联并传递记录。如果接收程序是由 EGL 生成的，则该记录用于初始化输入记录；如果接收程序是由 VisualAge Generator 生成的，则该记录用于初始化工作存储器。
<code>unloadTable (filename, selectStatement[, delimiter])</code>	将关系数据库中的数据卸装到文件中。
<code>verifyChkDigitMod10 (input, checkLength, result)</code>	验证以一系列整数开始的字符项中的模 10 校验数位。
<code>verifyChkDigitMod11 (input, checkLength, result)</code>	验证以一系列整数开始的字符项中的模 11 校验数位。
<code>wait (timeInSeconds0)</code>	将执行过程暂挂指定的秒数

beginDatabaseTransaction()

系统函数 **SysLib.beginDatabaseTransaction** 开始关系数据库事务，但仅当 EGL 运行时不会自动落实更改时才会如此。如果更改将自动落实，则该函数不起作用。

```
SysLib.beginDatabaseTransaction(  
    [database STRING in])
```

database

在 SysLib.connect 或 VGLib.connectionService 中指定的数据库名称。使用字符类型的文字或变量。

如果未指定连接，则函数将影响当前连接。

在调用 **SysLib.beginDatabaseTransaction** 时，事务在使用指定连接的下一次 I/O 操作时开始；并且事务会在发生落实或回滚时结束，如逻辑工作单元中所述。在落实或回滚后，EGL 运行时将恢复自动落实更改。

有关自动落实的详细信息，请参阅 SysLib.connect 和 sqlCommitControl。

相关概念

第 689 页的『EGL 函数的语法图』

第 279 页的『逻辑工作单元』

第 209 页的『SQL 支持』

相关参考

第 363 页的『sqlCommitControl』

第 820 页的『connect()』

第 839 页的『connectionService()』

bytes()

系统函数 **SysLib.bytes** 返回指定内存区的字节数。

```
SysLib.bytes(field fixedFieldOrArray in)  
returns (result INT)
```

result

一个数字项，它接收 *field* 中的字节数。有两种情况值得注意：

- 如果 *field* 是数组，则该函数返回一个元素中的字节数
- 如果 *field* 是 SQL 记录，则该函数返回记录中的字节数，包括额外的字节；有关详细信息，请参阅 SQL 记录时间间隔

field

数组、项或记录

Example():

```
result = SysLib.bytes(myItem);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 813 页的『EGL 库 SysLib』

第 31 页的『基本类型』

第 682 页的『SQL 记录内部结构』

calculateChkDigitMod10()

系统函数 **SysLib.calculateChkDigitMod10** 将模 10 校验数位放入以一系列整数开始的字符项中。

```
SysLib.calculateChkDigitMod10(  
    text anyChar inOut,  
    checkLength SMALLINT in,  
    result SMALLINT inOut)
```

text

以一系列整数开头的字符项。该项必须包含用于存放校验数位的附加位置，该位置紧跟在其它整数的右边。

checkLength

一个项，它包含 *text* 项中要使用的字符数，包括用于校验数位的位置。此项有 4 位，具有 SMALLINT 类型或 BIN 类型，并且不带小数位。

result

一个项，它接收下列两个值的其中一个：

- 0，如果已创建校验数位的话
- 1，如果未创建校验数位的话

此项有 4 位，具有 SMALLINT 类型或 BIN 类型，并且不带小数位。

可以在函数调用语句中使用 **SysLib.calculateChkDigitMod10**。

示例： 在以下示例中，myInput 是类型为 CHAR 的项并包含值 1734289；myLength 是类型为 SMALLINT 的项并包含值 7；myResult 是类型为 SMALLINT 的项：

```
SysLib.verifyChkDigitMod10 (myInput, myLength, myResult);
```

使用了一种算法来派生模 10 校验数位，在所有情况下，都不考虑校验数位位置中的数字。就示例值对算法描述如下：

1. 将输入数字的个位数位置乘以 2 并将每个备用位置从右到左乘以 2:

$$\begin{aligned}8 \times 2 &= 16 \\4 \times 2 &= 8 \\7 \times 2 &= 14\end{aligned}$$

2. 将乘积 (16814) 的各个位与未乘以 2 的输入数字位 (132) 相加:

$$1 + 6 + 8 + 1 + 4 + 1 + 3 + 2 = 26$$

3. 要得到校验数位，用以 0 结尾的下一个最大数字减去和数:

$$30 - 26 = 4$$

如果减运算得到 10，则校验数位是 0。

在此示例中，myInput 中的最前面的字符变为:

```
1734284
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 813 页的『EGL 库 SysLib』

calculateChkDigitMod11()

系统函数 **SysLib.calculateChkDigitMod11** 将模 11 校验数位放入以一系列整数开始的字符项中。

```
SysLib.calculateChkDigitMod11(  
    text anyChar inOut,  
    checkLength SMALLINT in,  
    result SMALLINT inOut)
```

text

以一系列整数开头的字符项。该项必须包含用于存放校验数位的附加位置，该位置紧跟在其它整数的右边。

checkLength

一个项，它包含 *text* 项中要使用的字符数，包括用于校验数位的位置。此项有 4 位，具有 SMALLINT 类型或 BIN 类型，并且不带小数位。

result

一个项，它接收下列两个值的其中一个：

- 0，如果已创建校验数位的话
- 1，如果未创建校验数位的话

此项有 4 位，具有 SMALLINT 类型或 BIN 类型，并且不带小数位。

可以在函数调用语句中使用 **SysLib.calculateChkDigitMod11**。

示例： 在以下示例中，myInput 是类型为 CHAR 的项并包含值 56621869；myLength 是类型为 SMALLINT 的项并包含值 8；myResult 是类型为 SMALLINT 的项：

```
SysLib.verifyChkDigitMod (myInput, myLength, myResult);
```

使用了一种算法来派生模 11 校验数位，在所有情况下，都不考虑校验数位位置中的数字。就示例值对算法描述如下：

1. 将输入数字的个位数位置中的数字乘以 2，在十位上乘以 3，在百位上乘以 4，依此类推，但将 myLength " 1 用作最大乘数；如果输入数字包含更多的位，则再次开始此序列并以 2 作为乘数：

```
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. 将第一个步骤得到的乘积累加并将和除以 11：

```
(12 + 24 + 4 + 10 + 36 + 42 + 10) / 11  
= 138 / 11  
= 12 余 6
```

3. 要得到校验数位，从 11 中减去余数以获得自检位：

```
11 - 6 = 5
```

如果余数是 0 或 1，则校验数位是 0。

在此示例中，myInput 中最前面的字符变为：

```
56621865
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 813 页的『EGL 库 SysLib』

callCmd()

系统函数 **SysLib.callCmd** 运行系统命令并等待命令完成。

```
SysLib.callCmd(  
    commandString STRING in  
    [, modeString STRING in  
    ])
```

commandString

标识要调用的操作系统命令。

modeString

modeString 可能是任何字符或字符串项。该项可能为下列任一方式：

- 表单：在表单方式中，输入的每个字符将按原样提供给程序，即，每次击键都会直接被传递给指定的命令。
- 行：在行方式中，直到使用换行符键之后，输入才会提供给程序，即，直到按 ENTER 键之后信息才会发送至指定命令，然后输入的整行将发送至命令。

要执行的系统命令对于正在运行的程序必须是可视的。例如，如果执行 **callCmd**("mySpecialProgram.exe")，程序“mySpecialProgram.exe”必须在环境变量 PATH 所指向的目录中。还可以指定完整的目录位置，例如，**callCmd**("program files/myWork/mySpecialProgram.exe")。

SysLib.callCmd 函数仅在 Java 环境中受支持。

使用 **SysLib.startCmd** 函数来运行系统命令并且不等待命令完成。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 813 页的『EGL 库 SysLib』

第 834 页的『startCmd()』

commit()

系统函数 **SysLib.commit** 保存上次落实之后对数据库和 MQSeries 消息队列所作的更新。生成的 Java 程序或包装器还保存由基于 CICS 的远程 COBOL 程序所作的更新（包括对 CICS 可恢复文件所作的更新），但仅当对远程 COBOL 程序进行的调用涉及客户机控制的工作单元时才这样做，如 *callLink* 元素中的 *luwControl* 所述。

```
SysLib.commit( )
```

在大多数情况下，EGL 执行依次影响每个可恢复的管理器的一阶段落实。但是，在 CICS for z/OS 上，**SysLib.commit** 导致 CICS SYNCPOINT，它将执行协调所有资源管理器的两阶段落实。

SysLib.commit 释放任何文件或数据库中的扫描位置和更新锁。

当将 **SysLib.commit** 与 MQ 记录配合使用时，下列描述是适用的：

- 仅当在 MQ 记录部件中选择了包括事务中的消息选项时才能恢复消息队列更新。
- 消息 **get** 和 **add** 都会受到可恢复消息的 **commit** 和 **rollback** 影响。如果在可恢复消息的 **get** 之后发出 **rollback**，则将把该消息放回到输入队列中，因此即使事务未能成功完成输入消息也不会丢失。同样，如果在可恢复消息的 **add** 之后发出 **rollback**，则将从队列中删除该消息。

可以通过避免毫无必要地使用 **SysLib.commit** 来提高性能。有关何时发生隐式落实的详细信息，请参阅逻辑工作单元。

示例：

```
sysLib.commit();
```

相关概念

第 689 页的『EGL 函数的语法图』

第 279 页的『逻辑工作单元』

第 242 页的『MQSeries 支持』

第 678 页的『运行单元』

第 209 页的『SQL 支持』

相关参考

第 845 页的『commitOnConverse』

第 848 页的『segmentedMode』

第 813 页的『EGL 库 SysLib』

第 377 页的『callLink 元素中的 luwControl』

第 561 页的『open』

第 574 页的『prepare』

conditionAsInt()

系统函数 **SysLib.conditionAsInt** 接受逻辑表达式（如 *myVar == 6*），如果表达式求值为 true，则返回 1，如果表达式求值为 false，则返回 0。

```
SysLib.conditionAsInt(logicalExpression AnyLogicalExpression in)  
returns (result SMALLINT)
```

result

类型为 SMALLINT 的值。

logicalExpression

一个逻辑表达式，如逻辑表达式中所述。

示例：

```
myField = -5;  
  
// result = 0  
result = SysLib.conditionAsInt(myField == 6);
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 813 页的『EGL 库 SysLib』

第 454 页的『逻辑表达式』

connect()

系统函数 **SysLib.connect** 允许程序在运行时连接数据库。此函数不返回值。

```
SysLib.connect(  
    database STRING in,  
    userID STRING in,  
    password STRING in  
    [, commitScope enumerationCommitScope in  
    [, disconnectOption enumerationDisconnectOption in  
    [, isolationLevel enumerationIsolationLevel in  
    [, commitControl enumerationCommitControlOption in  
    ]]] )
```

database

标识数据库:

- 将 *database* 设置为 **RESET** 将重新连接至缺省数据库, 但如果缺省数据库不可用, 则连接状态保持不变; 有关更多详细信息, 请参阅缺省数据库。
- 否则, 通过查找属性 **vgj.jdbc.database.server** 来查找物理数据库名称, 其中 *server* 是 **SysLib.connect** 调用所指定的数据库的名称。如果未定义此属性, 则按原样使用 **SysLib.connect** 调用所指定的数据库名称。
- 与非 J2EE 连接相比, J2EE 连接的数据库名称的格式有所不同:
 - 如果程序是为 J2EE 环境生成的, 则使用 JNDI 注册表中与数据源绑定的名称; 例如 jdbc/MyDB。如果构建描述符选项 **J2EE** 被设置为 **YES**, 就会发生这种情况。
 - 如果程序是为非 J2EE JDBC 环境生成的, 则使用连接 URL; 例如, jdbc:db2:MyDB。如果选项 **J2EE** 被设置为 **NO**, 就会发生这种情况。

userID

用来访问数据库的用户标识。此自变量必须是 **CHAR** 类型且长度为 8 的项, 文字也有效。此自变量是必需的。有关背景知识信息, 请参阅数据库权限和表名。

password

用来访问数据库的密码。此自变量必须是 **CHAR** 类型且长度为 8 的项, 文字也有效。此自变量是必需的。

commitScope

值是下列其中一个字, 不能使用引号, 也不能使用变量:

type1 (缺省值)

只支持一阶段落实。新连接将关闭所有游标、释放锁并结束任何现有连接; 然而, 在进行 **type1** 连接之前, 请调用 **SysLib.commit** 或 **SysLib.rollback**。

如果使用 **type1** 作为 *commitScope* 的值, 则参数 *disconnectOption* 的值必须是字 *explicit*, 这是缺省值。

type2

对数据库的连接不关闭游标、释放锁或结束现有连接。虽然可以使用多个连接来从多个数据库中进行读取, 但在一个工作单元中只应该更新一个数据库, 原因是只有一阶段落实可用。

twophase

等同于 **type2**。

disconnectOption

仅当正在生成 Java 输出时此参数才有意义。值是下列其中一个字, 不能使用引号, 也不能使用变量:

explicit (缺省值)

在程序调用 **SysLib.commit** 或 **SysLib.rollback** 之后, 连接保持活动。要释放连接资源, 程序必须发出 **SysLib.disconnect**。

如果使用 `type1` 作为 `commitScope` 的值, 则必须将参数 `disconnectOption` 的值设置为 (或允许其缺省为) 字 `explicit`。

automatic

落实或回滚将结束现有连接。

conditional

落实或回滚将自动结束现有连接, 除非已打开游标并且 `hold` 选项对该游标生效。有关 `hold` 选项的详细信息, 请参阅 `open`。

isolationLevel

此参数指示一个数据库事务相对于另一数据库事务的独立性级别。

下列字是按严格递增的顺序排列的, 与前面相同, 不能使用引号, 也不能使用变量:

- **readUncommitted**
- **readCommitted**
- **repeatableRead**
- **serializableTransaction** (缺省值)

有关详细信息, 请参阅 Sun 公司的 JDBC 文档。

commitControl

此参数指定在每次更改数据库之后是否落实。

有效值如下所示:

- **noAutoCommit** (缺省值) 表示落实不是自动进行的, 它通常会使执行速度更快。有关此情况下的落实和回滚的规则的信息, 请参阅逻辑工作单元。
- **autoCommit** 表示更新即时生效。

可临时将 `autoCommit` 切换为 `noAutoCommit`。有关详细信息, 请参阅 `SysLib.beginDatabaseTransaction`。

定义注意事项: **SysLib.connect** 设置下列系统变量:

- `VGVar.sqlerrd[3]`
- `SysVar.sqlca`
- `SysVar.sqlcode`
- `VGVar.sqlWarn[2]`

示例:

```
SysLib.connect(myDatabase, myUserId, myPassword);
```

相关概念

第 279 页的『逻辑工作单元』

第 678 页的『运行单元』

第 209 页的『SQL 支持』

相关任务

- 第 690 页的『EGL 语句和命令的语法图』
- 第 328 页的『设置 J2EE JDBC 连接』
- 第 240 页的『了解如何建立标准 JDBC 连接』

相关参考

- 第 425 页的『数据库权限和表名』
- 第 230 页的『缺省数据库』
- 第 813 页的『EGL 库 SysLib』

- 第 493 页的『Java 运行时属性（详细信息）』
- 第 561 页的『open』
- 第 364 页的『sqlDB』
- 第 815 页的『beginDatabaseTransaction()』
- 第 825 页的『disconnect()』
- 第 857 页的『sqlca』
- 第 858 页的『sqlcode』
- 第 870 页的『sqlerrd』
- 第 871 页的『sqlerrmc』
- 第 872 页的『sqlWarn』

convert()

系统函数 **SysLib.convert** 在 EBCDIC（主机）与 ASCII（工作站）格式之间转换数据，或者在单一格式内执行代码页转换。可将 **SysLib.convert** 用作函数调用语句中的函数名。

```
SysLib.convert(  
  target anyFixedItemOrRecordOrFormVariable inout,  
  direction enumerationConversionDirection in,  
  conversionTable CHAR(8) in)
```

target

具有您想要转换的格式的记录的、数据项或表单的名称。将根据目标对象中的最低级别项（不带子结构的项）的项定义来适当地转换数据。

在转换变长记录时，所转换的长度仅仅是当前记录的长度。当前记录的长度是使用记录的 `numElementsItem` 计算得到的，或者是在记录的 `lengthItem` 中设置的。如果变长记录在数字字段或 DBCHAR 字符的中间结束，则会发生转换错误，并且程序会结束。

direction

转换方向。有效值只有 "R" 和 "L"（包括引号）。如果指定了 `conversionTable`，则是必需的；否则是可选的。

"R" 缺省值。假定数据是远程格式，将把它转换为本地格式。

"L" 假定数据是本地格式，将把它转换为（转换表中定义的）远程格式。

conversionTable

数据项或文字（8 个字符，可选），它指定将要用于数据转换的转换表的名称。缺省值是与生成程序时指定的本地语言代码相关联的转换表。

定义注意事项： 可以使用链接选项部件来请求为远程调用生成自动数据转换、启动远程异步事务或者进行远程文件访问。总是使用对要进行转换的自变量定义的数据结构

来执行自动转换。如果自变量具有多种格式，则不要请求进行自动转换。或者，将程序编写为使用重新定义的记录声明（它正确地映射自变量的当前值）来显式地调用 **SysLib.convert**。

示例:

```
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
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 426 页的『数据转换』

第 813 页的『EGL 库 SysLib』

第 852 页的『callConversionTable』

defineDatabaseAlias()

系统函数 **SysLib.defineDatabaseAlias** 创建一个别名，该别名可用来建立与代码已连接至的数据库的新连接。建立连接后，可以下列任何函数中使用该别名：

- SysLib.connect
- SysLib.disconnect
- SysLib.beginDatabaseTransaction
- SysLib.setCurrentDatabase
- VGLib.connectionService

还可以在类型为 **ReportData** 的变量的 **connectionName** 字段中使用该别名。

```
SysLib.defineDatabaseAlias(
  alias STRING in,
  database STRING in)
```

alias

一个字符串文字或变量，充当第二个参数中标识的连接别名。别名是不区分大小写的。

database

在 `SysLib.connect` 或 `VGLib.connectionService` 中指定的数据库名称。使用字符类型的文字或变量。

如果未指定连接，则函数将影响当前连接。

示例如下所示:

```
// Connect to a database with alias "alias",
// which becomes the current connection.
defineDatabaseAlias( "alias", "database" );
connect( "alias", "user", "pwd" );

// Make two connections to the same database.
String db = "database";
defineDatabaseAlias( "alias1", db );
defineDatabaseAlias( "alias2", db );
connect( "alias1", "user", "pwd" );
connect( "alias2", "user", "pwd" );

// Another way to make two connections
// to the same database.
defineDatabaseAlias( "alias", "database" );
connect( "alias", "user", "pwd" );
connect( "database", "user", "pwd" );

// An alias is defined but not used. The second
// connect() does not create a new connection.
defineDatabaseAlias( "alias", "database" );
connect( "database", "user", "pwd" );
connect( "database", "user", "pwd" );

// Use of an alias (which is case-insensitive)
// when disconnecting.
defineDatabaseAlias( "alias", "database" );
connect( "aLiAs", "user", "pwd" );
disconnect( "ALIAS" );

// The next disconnect call fails because the
// connection is called "alias" not "database".
defineDatabaseAlias( "alias", "database" );
connect( "alias", "user", "pwd" );
disconnect( "database" );

// An alias may change. After the next call,
// "alias" refers to "firstDatabase"
defineDatabaseAlias( "alias", "firstDatabase" );

// After the next call,
// "alias" refers to "secondDatabase".
defineDatabaseAlias( "alias", "secondDatabase" );

// The last call would have failed
// if a connection was in place with "alias".
```

相关概念

第 689 页的『EGL 函数的语法图』

第 209 页的『SQL 支持』

相关参考

第 815 页的『beginDatabaseTransaction()』第 820 页的『connect()』
『disconnect()』
第 830 页的『setCurrentDatabase()』
第 839 页的『connectionService()』

disconnect()

系统函数 **SysLib.disconnect** 与指定的数据库断开连接，或者（如果未指定数据库的话）与当前数据库断开连接。

```
SysLib.disconnect(  
    [database STRING in  
    ])
```

database

在 SysLib.connect 或 VGLib.connectionService 中指定的数据库名称。使用字符类型的文字或变量。

在断开连接前，请调用 SysLib.commit 或 SysLib.rollback。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 813 页的『EGL 库 SysLib』
第 818 页的『commit()』
第 820 页的『connect()』
第 830 页的『rollback()』
第 839 页的『connectionService()』

disconnectAll()

系统函数 **SysLib.disconnectAll** 与当前连接的所有数据库断开连接。

在断开连接前，请调用 **SysLib.commit** 或 **SysLib.rollback**。

```
SysLib.disconnectAll( )
```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 813 页的『EGL 库 SysLib』
第 820 页的『connect()』
第 839 页的『connectionService()』

errorLog()

系统函数 **SysLib.errorLog** 将文本复制到由系统函数 **SysLib.startLog** 启动的错误日志中。

```
SysLib.errorLog(text STRING in)
```

text

要放在错误日志中的值。

日志条目包括编写该条目的日期和时间。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 813 页的『EGL 库 SysLib』

第 835 页的『startLog()』

getCmdLineArg()

系统函数 **SysLib.getCmdLineArg** 从用于调用 EGL 程序的自变量列表中返回指定自变量。指定自变量返回字符串值形式的自变量。

```
SysLib.getCmdLineArg(index INT in)  
returns (result STRING)
```

result

result 可能是任何字符项。

index

index 可能是任何整数项。

- 如果 *index* = 0, 将返回命令名。
- 如果 *index* = *n*, 将返回第 *n* 个自变量名。
- 如果 *n* 大于自变量计数, 将返回空格。

以下代码示例循环使用自变量列表:

```
count int;  
argument char(20);  
  
count = 0;  
argumentCount = SysLib.getCmdLineArgCount();  
  
while (count < argumentCount)  
    argument = SysLib.getCmdLineArg(count)  
    count = count + 1;  
end
```

SysLib.getCmdLineArg 函数仅在 Java 环境中受支持。

使用 **SysLib.getCmdLineArgCount** 函数来获取调用时传送至 EGL 主程序的自变量或参数的数目。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 813 页的『EGL 库 SysLib』

『getCmdLineArgCount()』

getCmdLineArgCount()

系统函数 **SysLib.getCmdLineArgCount** 返回用于启动 EGL 主程序的自变量的数目。

```
SysLib.getCmdLineArgCount( )  
returns (result INT)
```

result

result 是自变量的数目。

以下代码示例循环使用自变量列表:

```
count int;
argument char(20);

count = 0;
argumentCount = SysLib.getCmdLineArgCount();

while (count < argumentCount)
    argument = SysLib.getCmdLineArg(count)
    count = count + 1;
end
```

SysLib.getCmdLineArgCount 函数仅在 Java 环境中受支持。

使用 **SysLib.getCmdLineArg** 函数从 EGL 程序涉及的自变量列表中获取指定自变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 813 页的『EGL 库 SysLib』

第 826 页的『getCmdLineArg()』

getMessage()

系统函数 **SysLib.getMessage** 从 Java 运行时属性 `vgj.message.file` 中引用的文件中返回一条消息。可指定要包括在消息中的插入内容。在检索消息后，可在文本表单、打印表单、控制台表单、Web 页面或日志文件中显示它。

```
SysLib.getMessage(
    key STRING in
    [, insertArray STRING[] in])
returns (result STRING)
```

result

类型为 STRING 的字段。

key

类型为 STRING 的字符字段或文字。此参数提供进入在运行时使用的属性文件的键。如果键是空白的，则表示该消息是消息插入的并置。

insertArray

类型为 STRING 的数组。每个元素包含要包括在要检索的消息中的插入内容。

在消息文本中，替换符号是用花括号括起来的整数，如属性文件中的以下示例所示:

```
VGJ0216E = {0} is not a valid date mask for {1}.
```

insertArray 中的第一个元素被指定给编号为零的占位符，第二个元素被指定给编号为一的占位符，以此类推。

Java 运行时属性 `vgj.messages.file` 引用的文件的格式对于任何 Java 属性文件都是相同的。有关该格式的详细信息，请参阅程序属性文件。

相关概念

第 689 页的『EGL 函数的语法图』

第 315 页的『Java 运行时属性』

第 317 页的『程序属性文件』

相关参考

第 813 页的『EGL 库 SysLib』

第 493 页的『Java 运行时属性（详细信息）』

getProperty()

系统函数 **SysLib.getProperty** 检索 Java 运行时属性的值。如果找不到指定属性，则函数返回空字符串（""）。

```
SysLib.getProperty(propertyName STRING in)  
returns (result STRING)
```

result

类型为 **STRING** 的字段

propertyName

字符变量或常量，或者字符串文字

相关概念

第 689 页的『EGL 函数的语法图』

第 315 页的『Java 运行时属性』

相关参考

第 813 页的『EGL 库 SysLib』第 493 页的『Java 运行时属性（详细信息）』

loadTable()

系统函数 **SysLib.loadTable** 将文件中的数据装入到关系数据库中。

```
SysLib.loadTable(  
  fileName STRING in,  
  insertIntoClause STRING in  
  [, delimiter STRING in  
  ])
```

fileName

文件的名称。该名称是标准名称或是相对于从中调用该程序的目录的名称。

insertIntoClause

指定将提供数据的表和行。使用 SQL INSERT 语句中的 INSERT 子句的语法，如下示例中所示：

```
"INSERT INTO myTable(column1, column2)"
```

如果文件包括所有表列的值（按列顺序），则如下子句已经足够了：

```
"INSERT INTO myTable"
```

delimiter

指定在文件中隔开每个值的符号。（必须用换行符隔开每行数据。）

delimiter 的缺省符号是就是 Java 运行时属性 **vgj.default.databaseDelimiter** 中的值；而该属性的缺省值为竖杠（|）。

下列符号不可用：

- 十六进制字符 (0 到 9、a 到 f 和 A 到 F)
- 反斜杠 (\)
- 换行符或 *CONTROL-J*

要从关系数据库表中卸装信息并将其插入到文件中，使用 **SysLib.unloadTable** 函数。

相关参考

第 813 页的『EGL 库 SysLib』

第 493 页的『Java 运行时属性 (详细信息)』

第 836 页的『unloadTable()』

maximumSize()

系统函数 **SysLib.maximumSize** 返回动态数组中可以包含的最大行数；明确地说，此函数返回数组属性 **maxSize** 的值。

```
SysLib.maximumSize(arrayName anyArray in)
returns (result INT)
```

result 最大行数。

arrayName

动态数组的名称。

定义注意事项: 接收返回值的项必须具有 INT 类型，或者具有以下等效类型: BIN 类型，长度为 9 并且不带小数位。

可以通过包名和 / 或库名对数组名进行限定。

如果引用不是动态数组的项或记录，则会发生错误。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 68 页的『数组』

第 813 页的『EGL 库 SysLib』

queryCurrentDatabase()

系统函数 **SysLib.queryCurrentDatabase** 返回当前连接的数据库的产品号和发行版本号

```
SysLib.queryCurrentDatabase(
  product CHAR(8) inOut,
  release CHAR(8) inOut)
```

product

接收数据库产品名。此自变量必须是 CHAR 类型的项，且长度为 8。

要确定当代码连接至特定数据库时将接收到的字符串，请查看数据库或驱动程序的产品文档；或者在测试环境中运行代码并将接收到的值写入文件。

release

接收数据库发行版级别。此自变量必须是 CHAR 类型的项，且长度为 8。

要确定当代码连接至特定数据库时将接收到的字符串，请查看数据库或驱动程序的产品文档；或者在测试环境中运行代码并将接收到的值写入文件。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 813 页的『EGL 库 SysLib』

rollback()

系统函数 **SysLib.rollback** 撤销上次落实之后对数据库和MQSeries 消息队列所作的更新。该撤销发生在任何 EGL 生成的应用程序中。

```
SysLib.rollback( )
```

当程序由于错误状态而结束时，自动发生回滚。

定义注意事项: 当将 **SysLib.rollback** 与 MQ 记录配合使用时，下列描述是适用的:

- 仅当在 MQ 记录部件中选择了包括事务中的消息选项时才能恢复消息队列更新。
- 消息 **scan** 和 **add** 都受到可恢复消息的 **commit** 和 **rollback** 的影响。如果在可恢复消息的 **scan** 之后发出 **rollback**，则将把该消息放回到输入队列中，因此即使事务未能成功完成输入消息也不会丢失。同样，如果在可恢复消息的 **add** 之后发出 **rollback**，则将从队列中删除该消息。

目标平台:

平台	兼容性注意事项
iSeries、USS、Windows 2000 和 Windows NT	撤销对关系数据库和 MQSeries 消息队列所作的更改以及对使用客户机控制的工作单元调用的远程服务器程序所作的更改。

示例:

```
SysLib.rollback();
```

相关概念

第 689 页的『EGL 函数的语法图』

第 279 页的『逻辑工作单元』

第 242 页的『MQSeries 支持』

第 209 页的『SQL 支持』

相关参考

第 813 页的『EGL 库 SysLib』

setCurrentDatabase()

系统函数 **SysLib.setCurrentDatabase** 使指定的数据库成为当前活动数据库。

```
SysLib.setCurrentDatabase(database STRING in)
```

database

在 SysLib.connect 或 VGLib.connectionService 中指定的数据库名称。使用字符类型的文字或变量。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 813 页的『EGL 库 SysLib』

第 820 页的『connect()』

第 839 页的『connectionService()』

setError()

系统函数 **SysLib.setError** 将消息与 PageHandler 中的项相关联，或者与整个 PageHandler 相关联。该消息被放在 JSP 中的 JSF 消息或消息标记位置，并且是在相关 Web 页面显示时显示的。

如果验证函数调用 **SysLib.setError**，则当函数结束时将自动重新显示 Web 页面。

```
SysLib.setError(  
    itemInError anyPageItem in,  
    msgKey STRING in  
    {, itemInsert sysLibItemInsert in})
```

```
SysLib.setError(  
    this enumerationThis in,  
    msgKey STRING in  
    {, itemInsert sysLibItemInsert in})
```

```
SysLib.setError(msgText STRING in)
```

itemInError

出错的 PageHandler 项的名称。

this

指的是从中发出 **SysLib.setError** 的 PageHandler。在这种情况下，消息不是特定于项的，而是与整个 PageHandler 相关联。有关 **this** 的详细信息，请参阅对变量和常量的引用。

msgKey

字符项或文字（具有 CHAR 或 MBCHAR 类型），它提供在运行时使用的消息资源束或属性文件的键。如果键是空白的，则表示该消息是消息插入的并置。

itemInsert

作为输出消息插入而包括的字符项或文字。消息文本中的替换符号是用花括号括起来的整数，如以下示例所示：

文件名 {0} 无效

msgText

当未指定其它自变量时可以指定的字符项或文字。此文本与整个页相关联。

可以将多个消息与一个项或 PageHandler 相关联。当重新显示页时，EGL 运行时将显示消息。如果控制权被转发（明确地说，如果 PageHandler 运行 **forward** 语句），则那些消息将丢失。

相关概念

第 177 页的『PageHandler』

第 53 页的『引用 EGL 中的变量』

相关任务

第 690 页的『EGL 语句和命令的语法图』

相关参考

第 813 页的『EGL 库 SysLib』

第 531 页的『forward』

setLocale()

系统函数 **SysLib.setLocale** 用于 PageHandler 中。此函数设置 Java 语言环境，后者确定运行时行为的下列方面：

- 用于标号和消息的人类语言
- 缺省日期和时间格式

例如，可以在 Web 页面上显示语言列表，并根据用户的选择来设置 Java 语言环境。将使用新的 Java 语言环境，直到发生下列其中一种情况为止：

- 再次调用 **SysLib.setLocale**；或者
- 浏览器会话结束；或者
- 显示新的 Web 页面。

在提到的各种情况下，下一个 Web 页面还原（在缺省情况下）为浏览器中指定的 Java 语言环境。

如果用户提交表单或单击将要打开新窗口的链接，则原始窗口中的 Java 语言环境不受新窗口中的语言环境的影响。

SysLib.setLocale 符合 java.util.Locale 类的 JDK 1.1 和 1.2 API 文档。请参阅 ISO 639 以获取语言代码，请参阅 ISO 3166 以获取国家或地区代码。

```
SysLib.setLocale(  
    languageCode CHAR(2) in,  
    countryCode CHAR(2) in  
    [, variant CHAR(2) in])
```

languageCode

作为文字指定的或包含在 CHAR 类型的项中的两字符语言代码。仅 ISO 639 定义的语言代码有效。

countryCode

作为文字指定的或包含在 CHAR 类型的项中的两字符国家或地区代码。仅 ISO 3166 定义的国家或地区代码有效。

variant

一个变体，它是作为文字指定的或包含在 CHAR 类型的项中的两字符语言代码。此代码不是 Java 规范的一部分，但取决于浏览器和用户环境的其它方面。

相关概念

第 689 页的『EGL 函数的语法图』

第 177 页的『PageHandler』

相关参考第 813 页的『EGL 库 SysLib』

setRemoteUser()

系统函数 **SysLib.setRemoteUser** 设置调用远程程序时使用的用户标识和密码。

```
SysLib.setRemoteUser(  
    userID STRING in,  
    password STRING in)
```

userID

远程系统上的用户标识。

password

远程系统上的密码。

进行远程调用时，如果链接选项部件的 `callLink` 元素的属性 `remoteComType` 是 CICSJ2C、CICSECI 或 JAVA400，则权限基于传递给 **SysLib.setRemoteUser** 的值（如果是非空白的）。如果值为空白或未指定值，将在文件 **csouidpwd.properties** 中搜索该值，这包括属性 CSOUID（用于用户标识）和 CSOPWD（用于密码）。如果未使用这两种方法，则 EGL 运行时在没有用户名和密码的情况下进行调用。

在调用 **SysLib.setRemoteUser** 之前，代码会发出 Java 访问函数，这将显示一个对话框来提示用户输入用户标识和密码。在用户没有提供该信息时，可以将 **csouidpwd.properties** 中的一个值或两个值作为缺省值。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 343 页的『远程调用的 `csouidpwd.properties` 文件』

第 813 页的『EGL 库 `SysLib`』

第 381 页的『`callLink` 元素中的 `remoteComType`』

size()

系统函数 **SysLib.size** 返回指定的数据表中的行数或指定的数组中的元素数。该数组可以是结构项数组或数据项或记录的动态数组。

```
SysLib.size(arrayName anyArray in)  
returns (result INT)
```

result

指定的数据表中的行数或指定的数组中的元素数。

arrayName

数组或数据表的名称。

定义注意事项： 接收返回值的项必须具有 INT 类型，或者具有以下等效类型：BIN 类型，长度为 9 并且不带小数位。

如果数组名（*arrayName*）包含在另一个数组的子结构元素中，则返回的值是该结构项本身的出现次数，而不是包含结构的总出现次数（请参阅示例部分）。

可以通过包名和 / 或库名对数组名进行限定。

如果引用不是数组的项或记录，则会发生错误。

示例： 此示例使用 **SysLib.size** 返回的值来控制循环：

```
// Calculate the sum of an array of numbers  
sum = 0;  
i = 1;  
myArraySize = SysLib.size(myArray);
```

```

while (i <= myArraySize)
    sum = myArray[i] + sum;
    i = i + 1;
end

```

下面，请参照以下记录部件：

```

Record myRecordPart
    10 siTop CHAR(40)[3];
    20 siNext CHAR(20)[2];
end

```

假定根据 `myRecordPart` 来创建记录，则可以使用 **SysLib.size(siNext)** 来确定下级数组的 `occurs` 值：

```

// Sets count to 2
count = SysLib.size(myRecord.siNext);

```

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 68 页的『数组』

第 813 页的『EGL 库 SysLib』

startCmd()

系统函数 **SysLib.startCmd** 运行系统命令并且不等待命令完成。

```

SysLib.startCmd(
    commandString STRING in
    [, modeString STRING in
    ])

```

commandString

标识要调用的操作系统命令。

modeString

modeString 可能是任何字符或字符串项。该项可能为下列任一方式：

- 表单：在表单方式中，输入的每个字符将按原样提供给程序，即，每次击键都会直接被传递给指定的命令。
- 行：在行方式中，直到使用换行符之后，输入才会提供给程序，即，直到按 ENTER 键之后信息才会发送至指定命令，然后输入的整行将发送至命令。

以下代码示例

```

example from Arlan here...

```

要执行的系统命令对于正在运行的程序必须是可视的。例如，如果执行 **callCmd("mySpecialProgram.exe")**，程序“mySpecialProgram.exe”必须在环境变量 `PATH` 所指向的目录中。还可以指定完整的目录位置，例如，**callCmd("program files/myWork/mySpecialProgram.exe")**。

SysLib.startCmd 函数仅在 Java 环境中受支持。

使用 **SysLib.callCmd** 函数来运行系统命令并等待命令完成。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 813 页的『EGL 库 SysLib』

必要时进行哑元更改

startLog()

系统函数 **SysLib.startLog** 用来打开错误日志。每当程序调用 **SysLib.errorLog** 时，就会将文本写入到该日志中。

```
SysLib.startLog(logFile STRING in)
```

logFile

错误日志。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 813 页的『EGL 库 SysLib』

第 825 页的『errorLog()』

startTransaction()

系统函数 **SysLib.startTransaction** 以异步方式调用主程序，将该程序与打印机或终端设备相关联并传递记录。如果接收程序是由 EGL 生成的，则该记录用于初始化输入记录；如果接收程序是由 VisualAge Generator 生成的，则该记录用于初始化工作存储器。

此函数的缺省行为是启动位于同一个 Java 包中的程序。要更改此行为，请在用来生成调用程序的链接选项部件中指定 `asynchLink` 元素。

Java 程序只能转移至同一台机器上的另一个 Java 程序。

```
SysLib.startTransaction(  
  request anyBasicRecord in  
  [, prID startTransactionPrId in  
  [, termID CHAR(4) in ]])
```

request

基本记录的名称，该记录必须具有以下格式：

- 前 2 个字节（具有 `SMALLINT` 类型或具有不带小数的 `BIN` 类型）包含要传递给已启动事务的数据的长度加上 10（用于两个不传递的字段，包括此字段）。
- 接下来的 8 个字节（具有 `CHAR` 类型）也不会被传递，但它们包含将要启动的程序的名称。
- 将传递请求记录的其余部分。

prID

这个 4 字节项会被忽略（如果指定的话）。

termID

这个具有 `CHAR` 类型的 4 字节项会被忽略（如果指定的话）。如果指定 *termID*，则必须指定 *prID*。

相关概念

第 689 页的『EGL 函数的语法图』

相关参考

第 343 页的『asynchLink 元素』

第 813 页的『EGL 库 SysLib』

第 853 页的『errorCode』

第 846 页的『printerAssociation』

第 589 页的『transfer』

unloadTable()

系统函数 **SysLib.unloadTable** 将关系数据库中的数据卸装到文件中。

```
SysLib.unloadTable(  
    fileName STRING in,  
    selectStatement STRING in  
    [, delimiter STRING in  
    ])
```

fileName

文件的名称。该名称是标准名称或是相对于从中调用该程序的目录的名称。

selectStatement

指定从关系数据库中选择数据的条件。使用不包括主变量的 SQL SELECT 语句的语法，例如：

```
"SELECT column1, column2 FROM myTABLE  
WHERE column3 > 10"
```

delimiter

指定在文件中隔开每个值的符号。（必须用换行符隔开每行数据。）

delimiter 的缺省符号就是 Java 运行时属性 **vgj.default.databaseDelimiter** 中的值；而该属性的缺省值为竖杠（|）。

下列符号不可用：

- 十六进制字符（0 到 9、a 到 f 和 A 到 F）
- 反斜杠（\）
- 换行符或 *CONTROL-J*

要从文件中装入信息并将其插入到关系数据库表中，使用 **SysLib.loadTable** 函数。

相关参考

第 813 页的『EGL 库 SysLib』

第 828 页的『loadTable()』

第 493 页的『Java 运行时属性（详细信息）』

verifyChkDigitMod10()

系统函数 **SysLib.verifyChkDigitMod10** 验证以一系列整数开始的字符项中的模 10 校验数位。

```
SysLib.verifyChkDigitMod10(  
    text anyChar in,  
    checkLength SMALLINT in,  
    result SMALLINT inOut)
```

text

以一系列整数开头的字符项。该项包含用于存放校验数位的附加位置，该位置紧跟在其它整数的右边。

checkLength

一个项，它包含 *text* 项中要使用的字符数，包括用于校验数位的位置。此项有 4 位，具有 SMALLINT 类型或 BIN 类型，并且不带小数位。

result

一个项，它接收下列两个值的其中一个：

- 0，如果计算得到的校验数位与 *text* 中的值相匹配的话
- 1，如果计算得到的校验数位与该值不匹配的话

此项有 4 位，具有 SMALLINT 类型或 BIN 类型，并且不带小数位。

可以在函数调用语句中使用 **SysLib.verifyChkDigitMod10**；也可将其用作文本表单中的项验证器。

示例： 在以下示例中，myInput 是类型为 CHAR 的项并包含值 1734284；myLength 是类型为 SMALLINT 的项并包含值 7；myResult 是类型为 SMALLINT 的项：

```
SysLib.verifyChkDigitMod10 (myInput, myLength, myResult);
```

用来派生模 10 校验数位的算法，在所有情况下，不考虑校验数位位置上的数；但当算法完成后，计算的值与校验数位位置上的数进行比较。

就示例值对算法描述如下：

1. 将输入数字的个位数位置乘以 2 并将每个备用位置从右到左乘以 2:

$$\begin{aligned}8 \times 2 &= 16 \\4 \times 2 &= 8 \\7 \times 2 &= 14\end{aligned}$$

2. 将乘积 (16814) 的各个位与未乘以 2 的输入数字位 (132) 相加:

$$1 + 6 + 8 + 1 + 4 + 1 + 3 + 2 = 26$$

3. 要得到校验数位，用以 0 结尾的下一个最大数字减去和数:

$$30 - 26 = 4$$

如果减运算得到 10，则校验数位是 0。

在此示例中，计算得到的校验数位与校验数位位置中的值相匹配，并且 myResult 的值是 0。

相关参考

第 813 页的『EGL 库 SysLib』

第 62 页的『验证属性』

verifyChkDigitMod11()

系统函数 **SysLib.verifyChkDigitMod11** 验证以一系列整数开始的字符项中的模 11 校验数位。

```
SysLib.verifyChkDigitMod11(  
text anyChar in,  
checkLength SMALLINT in,  
result SMALLINT inOut)
```

text

以一系列整数开头的字符项。该项包含用于存放校验数位的附加位置，该位置紧跟在其它整数的右边。

checkLength

一个项，它包含 *text* 项中要使用的字符数，包括用于校验数位的位置。此项有 4 位，具有 SMALLINT 类型或 BIN 类型，并且不带小数位。

result

一个项，它接收下列两个值的其中一个：

- 0，如果计算得到的校验数位与 *text* 中的值相匹配的话
- 1，如果计算得到的校验数位与该值不匹配的话

此项有 4 位，具有 SMALLINT 类型或 BIN 类型，并且不带小数位。

可以在函数调用语句中使用 **SysLib.verifyChkDigitMod11**；也可将其用作文本表单中的项验证器。

示例： 在以下示例中，myInput 是类型为 CHAR 的项并包含值 56621869；myLength 是类型为 SMALLINT 的项并包含值 8；myResult 是类型为 SMALLINT 的项：

```
sysLib.verifyChkDigitMod11 (myInput, myLength, myResult);
```

使用了一种算法来派生模 11 校验数位，在所有情况下，都不考虑校验数位位置中的数字；但是，当算法完成时，将计算得到的值与校验数位位置中的数字作比较。就示例值对算法描述如下：

1. 将输入数字的个位数位置中的数字乘以 2，在十位上乘以 3，在百位上乘以 4，依此类推，但将 myLength " 1 用作最大乘数；如果输入数字包含更多的位，则再次开始此序列并以 2 作为乘数：

```
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. 将第一个步骤得到的乘积累加并将和除以 11：

```
(12 + 24 + 4 + 10 + 36 + 42 + 10) / 11
= 138 / 11
= 12 余 6
```

3. 要得到校验数位，从 11 中减去余数以获得自检位：

```
11 - 6 = 5
```

如果余数是 0 或 1，则校验数位是 0。

在此示例中，计算得到的校验数位与校验数位位置中的值相匹配，并且 myResult 的值是 0。

相关参考

第 813 页的『EGL 库 SysLib』

第 62 页的『验证属性』

wait()

系统函数 **SysLib.wait** 将执行过程暂挂指定的秒数。

```
SysLib.wait(timeInSeconds BIN(9,2) in)
```

timeInSeconds

时间可以是任何数字项或文字。如果数字不是整数，则秒的有效小数部分可达百分之一秒。

当两个未同时运行的程序需要通过共享文件或数据库中的记录进行通信时，可以使用 **SysLib.wait**。一个程序可能需要暂停处理，直到另一个程序更新共享记录中的信息为止。

示例

```
SysLib.wait(15); // waits for 15 seconds
```

相关概念 第 689 页的『EGL 函数的语法图』

相关参考

第 813 页的『EGL 库 SysLib』

EGL 库 VGLib

VGLib 函数如下所示:

系统函数 / 调用	描述
<code>connectionService(<i>userID</i>, <i>password</i>, <i>serverName</i> [, <i>product</i>, <i>release</i> [, <i>connectionOption</i>]])</code>	它有两个优点: <ul style="list-style-type: none">• 允许程序在运行时连接数据库或与数据库断开连接。• (可选)接收数据库产品名和发行版级别。可以在 case、if 或 while 语句中使用接收到的信息，以使运行时处理取决于数据库的特征。
<code>result = getVAGSysType()</code>	标识要运行程序的目标系统。

connectionService()

系统函数 **VGLib.connectionService** 有两个优点:

- 允许程序在运行时连接数据库或与数据库断开连接。
- (可选)接收数据库产品名和发行版级别。可以在 **case**、**if** 或 **while** 语句中使用接收到的信息，以使运行时处理取决于数据库的特征。

当使用 **VGLib.connectionService** 来从 Java 程序中创建新连接时，通过设置系统变量 **VGVar.sqlIsolationLevel** 来指定隔离级别。

VGLib.connectionService 仅用于从 VisualAge Generator 和 EGL 5.0 迁移的程序。如果 (在开发时) 选择了 EGL 首选项 **VisualAge Generator 兼容性** 或者 (在生成时) 将构建描述符选项 **VAGCompatibility** 设置为 *yes*，则支持此函数。

对于新程序，改为使用下列系统函数:

- SysLib.connect

- SysLib.disconnect
- SysLib.disconnectAll
- SysLib.queryCurrentDatabase
- SysLib.setCurrentDatabase

VGLib.connectionService 不返回值。

```
VGLib.connectionService(
  userID CHAR(8) in,
  password CHAR(8) in,
  serverName CHAR(18) in
  [, product CHAR(8) inOut,
   release CHAR(8) inOut
  [, connectionOption STRING in
  ]])
```

userID

用来访问数据库的用户标识。此自变量必须是 CHAR 类型的项，且长度为 8；文字无效。此自变量是必需的。有关背景知识信息，请参阅数据库权限和表名。

password

用来访问数据库的密码。此自变量必须是 CHAR 类型的项，且长度为 8；文字无效。此自变量是必需的。

serverName

指定一个连接并使用该连接来对自变量 *product* 和 *release* 赋值（如果在 **VGLib.connectionService** 的调用中包括了那些自变量的话）。

自变量 *serverName* 是必需的，它必须是 CHAR 类型的项，并且长度为 18。下列任何值都有效：

空白（无内容）

如果有适当的连接，则 **VGLib.connectionService** 将保留该连接。如果没有适当的连接，则结果（不同于赋值）是返回到运行单元开始时起作用的连接状态，如缺省数据库中所述。

RESET

RESET 重新连接至缺省数据库；但如果缺省数据库不可用，则连接状态保持不变。

有关更多详细信息，请参阅缺省数据库。

serverName

标识数据库：

- 通过查找属性 **vgj.jdbc.database.server** 来查找物理数据库名称，其中 *server* 是 **VGLib.connectionService** 调用所指定的服务器的名称。如果未定义此属性，则按原样使用 **VGLib.connectionService** 调用所指定的服务器名称。
- 与非 J2EE 连接相比，J2EE 连接的数据库名称的格式有所不同：
 - 如果程序是为 J2EE 环境生成的，则使用 JNDI 注册表中与数据源绑定的名称；例如 jdbc/MyDB。如果构建描述符选项 **J2EE** 被设置为 YES，就会发生这种情况。
 - 如果程序是为非 J2EE JDBC 环境生成的，则使用连接 URL；例如，jdbc:db2:MyDB。如果选项 **J2EE** 被设置为 NO，就会发生这种情况。

product

接收数据库产品名。此自变量（如果指定的话）必须是 CHAR 类型的项，且长度为 8。

要确定当代码连接至特定数据库时将接收到的字符串，请查看数据库或驱动程序的产品文档；或者在测试环境中运行代码并将接收到的值写入文件。

release

接收数据库发行版级别。此自变量（如果指定的话）必须是 CHAR 类型的项，且长度为 8。

要确定当代码连接至特定数据库时将接收到的字符串，请查看数据库或驱动程序的产品文档；或者在测试环境中运行代码并将接收到的值写入文件。

connectionOption

有效值如下所示：

D1E

D1E 是缺省值。选项名中的 *I* 指示仅支持一阶段落实，*E* 指示任何断开连接都必须是显式的。在这种情况下，落实或回滚对现有连接不起作用。

对数据库的连接不关闭游标、释放锁或结束现有连接。但是，如果运行单元已连接至同一数据库，则效果等同于指定 DISC 并接着指定 D1E。

可以使用多个连接来从多个数据库进行读取，但是，由于只有一阶段落实可用，所以在一个工作单元中只应该更新一个数据库。

D1A

选项名中的 *I* 指示仅支持一阶段落实，*A* 指示任何断开连接都是自动的。此选项的特征如下所示：

- 每次只能连接至一个数据库
- 对数据库的落实、回滚或连接将结束现有的连接

DISC

与指定的数据库断开连接。与数据库断开连接将导致回滚并释放锁，但仅适用于该数据库。

DCURRENT

与当前已连接的数据库断开连接。与数据库断开连接将导致回滚并释放锁，但仅适用于该数据库。

DALL

与所有已连接的数据库断开连接。与所有数据库断开连接将导致对那些数据库执行回滚，但不对其余可恢复的资源执行回滚。

SET

设置当前连接。（缺省情况下，在运行单元中最近进行的连接就是当前连接。）

为了与 VisualAge Generator 兼容，支持下列值，但这些值等同于 D1E: R、D1C、D2A、D2C 和 D2E。

定义注意事项: **VGLib.connectionService** 设置下列系统变量：

- VGVar.sqlerrd
- SysVar.sqlca
- SysVar.sqlcode

- `VGVar.sqlWarn`

示例:

```
VGLib.connectionService(myUserId, myPassword,
    myServerName, myProduct, myRelease, "D1E");
```

相关概念

- 第 689 页的『EGL 函数的语法图』
- 第 279 页的『逻辑工作单元』
- 第 678 页的『运行单元』
- 第 209 页的『SQL 支持』

相关任务

- 第 690 页的『EGL 语句和命令的语法图』
- 第 328 页的『设置 J2EE JDBC 连接』
- 第 240 页的『了解如何建立标准 JDBC 连接』

相关参考

- 第 425 页的『数据库权限和表名』
- 第 230 页的『缺省数据库』
- 第 839 页的『EGL 库 VGLib』
- 第 493 页的『Java 运行时属性 (详细信息)』
- 第 364 页的『sqlDB』
- 第 857 页的『sqlca』
- 第 858 页的『sqlcode』
- 第 870 页的『sqlerrd』
- 第 871 页的『sqlerrmc』
- 第 872 页的『sqlIsolationLevel』
- 第 872 页的『sqlWarn』

getVAGSysType()

系统函数 **VGLib.getVAGSysType** 标识正在运行程序的目标系统。如果（在开发时）选择了程序属性 **VAGCompatibility** 或者（在生成时）将构建描述符选项 **VAGCompatibility** 设置为 *yes*，则支持此函数。

如果生成的输出是 Java 包装器，则 **VGLib.getVAGSysType** 不可用。否则，此函数返回 VisualAge Generator EZESYS 特殊函数数字已返回的字符值。如果当前系统不受 VisualAge Generator 支持，则此函数返回 **SysVar.systemType** 所返回的代码的大写等效字符串。

```
VGLib.getVAGSysType( )
returns (result CHAR(8))
```

result

一个字符串，它包含系统类型代码，如下表所示。

VGLib.getVAGSysType 返回 **SysVar.systemType** 的值的 VisualAge Generator 等效值。

sysVar.systemType 中的值	VGLib.getVAGSysType 返回的值
AIX	"AIX"

sysVar.systemType 中的值	VGLib.getVAGSysType 返回的值
DEBUG	"ITF"
ISERIESC	"OS400"
ISERIESJ	"OS400"
LINUX	"LINUX"
USS	"OS390"
WIN	"WINNT"

只能将 **VGLib.getVAGSysType** 返回的值用作字符串；不能将返回的值与逻辑表达式中的操作数 *is* 或 *not* 配合使用（对于 **sysVar.systemType**，这是可以的）：

```
// valid ONLY for sysVar.systemType
if sysVar.systemType is AIX
  call myProgram;
end
```

只能将 **VGLib.getVAGSysType** 用作赋值或 **move** 语句中的源。

VGLib.getVAGSysType 的特征如下所示：

基本类型

CHAR

数据长度

8（用空格填充）

在转换之后是否始终将值恢复？

是

建议您使用 **sysVar.systemType** 而不是 **VGLib.getVAGSysType**。

定义注意事项： **VGLib.getVAGSysType** 的值不影响在生成时验证的代码。例如，即使正在为 Windows 进行生成，也验证以下 **add** 语句：

```
mySystem CHAR(8);
mySystem = VGLib.getVAGSysType();
if (mySystem == "AIX")
  add myRecord;
end
```

为了避免验证永远不会在目标系统中运行的代码，可以将不想验证的语句移至第二个程序；然后，让原始程序有条件地调用新程序：

```
mySystem CHAR(8);
mySystem = VGLib.getVAGSysType();

if (mySystem == "AIX")
  call myAddProgram myRecord;
end
```

也可以采用另一种方法来解决这个问题，但仅当使用 **sysVar.systemType** 而不是 **VGLib.getVAGSysType** 时该方法才可行；有关详细信息，请参阅 *eliminateSystemDependentCode*。

相关参考

第 839 页的『EGL 库 VGLib』

第 354 页的『eliminateSystemDependentCode』

第 859 页的『systemType』

EGL 库外部的系统变量

包含在 EGL 库中的变量对于运行单元是全局的。其它系统变量具有不同的作用域特征并且分类如下：

ConverseVar

主要在 textUI 应用程序中使用的变量。

SysVar

用于一般用途的变量。

VGVar

主要用于从 VisualAge Generator 迁移的应用程序的变量。

如果在作用域中已经有另一个系统变量具有同名标识的情况下引用该系统变量，则必须包括类别名作为限定符。例如，如果作用域中有另一个变量名为 **eventKey**，则必须指定 **ConverseVar.eventKey** 而不是 **eventKey**。如果同名标识不在作用域中，则限定符是可选的。

相关概念

第 53 页的『引用 EGL 中的变量』

第 52 页的『EGL 中的确定作用域规则和“this”』

相关参考

『ConverseVar』

第 849 页的『SysVar』

第 862 页的『VGVar』

ConverseVar

限定符 **ConverseVar** 可以放在下表中列出的每个 EGL 系统变量的名称的前面。这些变量主要在 textUI 应用程序中使用。

系统变量	描述
commitOnConverse	指定是否在非分段程序发出 converse 之前在文本应用程序中落实并释放资源。对于非分段程序，缺省值为 0（表示 no），对于分段程序，缺省值为 1（表示 yes）。
eventKey	标识一个键，用户按下该键以将表单返回至 EGL 文本程序。
printerAssociation	允许您在运行时打印一个打印表单时指定输出目标。
segmentedMode	在文本应用程序中用于更改 converse 语句的效果，但在被调用程序中，忽略用于此用途的此变量。

系统变量	描述
validationMsgNum	包含文本应用程序中的 ConverseLib.validationFailed 赋予的值，所以您可以确定验证函数是否报告错误。

相关概念

第 53 页的『引用 EGL 中的变量』

第 52 页的『EGL 中的确定作用域规则和“this”』

相关参考

第 844 页的『EGL 库外部的系统变量』

commitOnConverse

系统变量 **ConverseVar.commitOnConverse** 指定在非分段程序发出 `converse` 之前是否在文本应用程序中落实并释放资源。对于非分段程序，缺省值为 0（表示 *no*），对于分段程序，缺省值为 1（表示 *yes*）。

可以按照下列任何方式使用 **ConverseVar.commitOnConverse**:

- 作为赋值语句或 **move** 语句的源或目标
- 作为在 **case** 语句、**if** 语句或 **while** 语句中使用的逻辑表达式中的变量
- 作为 **return** 语句或 **exit** 语句中的自变量

ConverseVar.commitOnConverse 的其它特征如下所示:

基本类型

NUM

数据长度

1

在转换之后是否始终将值恢复?

是

有关使用此变量的详细信息，请参阅分段。

相关概念

第 147 页的『文本应用程序中的分段』

相关参考

第 519 页的『`converse`』

第 844 页的『EGL 库外部的系统变量』

eventKey

系统变量 **ConverseVar.eventKey** 标识一个键，用户按下该键来将表单返回至 EGL 文本或程序。程序每次运行 **converse** 语句时都会将此值重置。

如果 EGL 代码没有输入表单，则 **ConverseVar.eventKey** 的初始值是 **ENTER**。

下列值有效（无论是大写的、小写的还是组合大小写的）:

- **ENTER**

- **BYPASS**（它指的是任何作为表单的旁路键指定的键；或者，如果未对表单指定任何旁路键，则它指的是任何作为 `formGroup` 的旁路键指定的键；或者，如果未对 `formGroup` 指定任何旁路键，则它指的是任何作为程序的旁路键指定的键）
- **PA1 至 PA3**
- **PF1 至 PF24**（也用于 F1 至 F24）
- **PAKEY**（表示任何 PA 键）
- **PFKEY**（表示任何 PF 或 F 键）

注：PA 键总是被认为是旁路键。

可以将 `ConverseVar.eventKey` 用作 `if` 或 `while` 语句中的操作数。

此系统变量的特征如下所示：

基本类型

CHAR

数据长度

1

跨段保存值

否

`ConverseVar.eventKey` 在批处理程序中无效。

示例：`ConverseVar.eventKey` 的比较运算符是 `is` 或 `not`，如以下示例所示：

```
if (ConverseVar.eventKey IS PF3)
  exit program(0);
end
```

相关参考

第 454 页的『逻辑表达式』

第 844 页的『EGL 库外部的系统变量』

printerAssociation

系统变量 `ConverseVar.printerAssociation` 允许您在运行时打印一个打印表单时指定输出目标。

可以按照下列任何方式来使用此变量：

- 作为赋值语句或 `move` 语句中的源或目标
- 作为逻辑表达式中的比较值
- 作为 `return` 语句中的值

`ConverseVar.printerAssociation` 的特征如下所示：

基本类型

CHAR

数据长度

随文件类型的不同而有所变化

在转换之后是否始终将值恢复？

是

ConverseVar.printerAssociation 被初始化为生成或调试期间指定的系统资源名称。如果程序将控制权传递给另一个程序，则 **ConverseVar.printerAssociation** 的值将被重置为接收程序的缺省值。

即使给定的打印表单允许多个打印作业，`close` 语句也仅关闭与 **ConverseVar.printerAssociation** 的当前值相关的文件。

特定于 **Java** 输出的详细信息：对于 **Java** 输出，您将 **ConverseVar.printerAssociation** 设置为分为两个部分的字符串，这两部分之间有一个冒号：

jobID:destination

jobID 一个字符序列（不带冒号），它唯一地标识每个打印作业。这些字符区分大小写（*job01* 与 *JOB01* 不同），在打印作业关闭后，可以重用 *jobID*。

根据代码中的事件流的不同，可以使用不同的作业来转储另一种类型的输出或另一输出顺序。例如，请参照以下 EGL 语句序列：

```
ConverseVar.printerAssociation = "job1";
print form1;
ConverseVar.printerAssociation = "job2";
print form2;
ConverseVar.printerAssociation = "job1";
print form3;
```

当程序结束时，将创建两个打印作业：

- form1，后跟 form3
- form2，单独

destination

接收输出的打印机或文件。

字符串 *destination* 是可选的，如果打印作业仍处于打开状态，则该字符串将被忽略。如果不存在该字符串，则下列描述是适用的：

- 可以省略 *destination* 前面的冒号
- 在大多数情况下，程序会显示一个打印预览对话框，从该对话框中，用户可以指定用于输出的打印机或文件。在 UNIX 上使用 `curses` 库的情况例外；在该情况下，打印作业将转至缺省打印机。

当正在为 Windows 2000/NT/XP 进行生成时，下列情况适用于 *destination* 的设置：

- 要将输出发送至缺省打印机，执行下列操作：
 - 指定与资源关联部件中的 **fileName** 属性相匹配的值。
 - 更改 Java 运行时属性，使相关文件类型的值是 *spool*（而不是 *seqws*）。例如，在资源关联部件中，如果 **fileName** 属性的值是 *myFile*，并且 **systemName** 的值是 *printer*，则必须更改 Java 运行时属性的设置，以使 `vgj.ra.myFile.fileType` 设置为 *spool* 而不是 *seqws*。在进行更改之后，属性如下所示：

```
vgj.ra.myFile.systemName=printer
vgj.ra.myFile.fileType=spool
```

- 要将输出发送到一个文件，请指定与资源关联部件中的 **fileName** 属性相匹配的值（当资源关联部件中的相关 **fileType** 属性的值是 *seqws* 时）。**systemName** 属性是指一个资源关联部件，它包含接收输出的操作系统文件的名称。
- 不要指定 *printer* 值来作为 *destination* 的值。如果这样做的话，将向用户显示打印预览对话框，但在以后的 EGL 版本中可能会更改该行为。

当正在为 UNIX 进行生成时，下列情况适用于 *destination* 的设置：

- 要将输出发送至缺省打印机（而不考虑是否正在使用 *curses* 库），请指定与资源关联部件中的 **fileName** 属性相匹配的值（当资源关联部件中的相关 **fileType** 属性的值是 *spool* 时）。
- 要将输出发送到一个文件，请指定与资源关联部件中的 **fileName** 属性相匹配的值（当资源关联部件中的相关 **fileType** 属性的值是 *seqws* 时）。资源关联部件中的 **systemName** 属性包含接收输出的操作系统文件的名称。
- 不要指定 *printer* 值来作为 *destination* 的值。如果这样做的话（并且如果未使用 *curses* 库的话），将向用户显示打印预览对话框，但在以后的 EGL 版本中可能会更改该行为。

segmentedMode

在文本应用程序中，使用系统变量 **ConverseVar.segmentedMode** 来更改 *converse* 语句的效果，但在被调用程序中，忽略用于此用途的此变量。有关背景知识信息，请参阅分段。

ConverseVar.segmentedMode 的值如下所示：

- 1** 下一个 **converse** 语句以分段方式运行。
- 0** 下一个 **converse** 语句以非分段方式运行。

对于非分段程序，缺省值为 0，对于分段程序，缺省值为 1。在运行 **converse** 语句后，此变量被重置为缺省值。

可以按照下列任何方式来使用此变量：

- 作为赋值语句或 *move* 语句中的源或目标
- 作为 **move...for count** 语句中的计数值
- 作为逻辑表达式中的比较值
- 作为 *return* 语句中的值

ConverseVar.segmentedMode 的特征如下所示：

基本类型

NUM

数据长度

1

在转换之后是否将值恢复？

否

相关概念

第 147 页的『文本应用程序中的分段』

相关参考

第 844 页的『EGL 库外部的系统变量』

validationMsgNum

系统变量 **ConverseVar.validationMsgNum** 包含文本应用程序中的 **ConverseLib.validationFailed** 赋予的值，所以您可以确定验证函数是否报告错误。在下列每一种情况下，值被重置为零：

- 程序初始化
- 程序发出 `converse`、`display` 或 `print` 语句
- 作为验证错误的结果，程序重新发出 `converse` 语句以显示文本表单

可以按照下列方式使用 **ConverseVar.validationMsgNum**：

- 作为赋值语句或 **move** 语句的源或目标（在 **move** 语句的“for count”中也是允许的）
- 作为逻辑表达式中的变量
- 作为 **return** 语句或 **exit** 语句中的自变量

ConverseVar.validationMsgNum 的特征如下所示：

基本类型

INT

在转换之后是否始终将值恢复？

否

示例

```
/*Keep the first message number that was set
during validation routines */
if (ConverseVar.validationMsgNum > 0)
  ConverseLib.validationFailed(10);
end
```

相关参考

第 519 页的『`converse`』

第 722 页的『`validationFailed()`』

第 522 页的『`display`』

第 576 页的『`print`』

第 844 页的『EGL 库外部的系统变量』

SysVar

限定符 **SysVar** 可以放在下表中列出的每个 EGL 系统变量的名称的前面。这些变量用于一般用途。

系统变量	描述
arrayIndex	包含数字: <ul style="list-style-type: none"> • 数组中的第一个与搜索条件相匹配的元素的编号, 该搜索条件是带有 in 运算符的简单逻辑表达式。 • 零, 如果没有任何数组元素与搜索条件相匹配的话。 • 在 move ... for count 语句之后, 在目标数组中修改的最后一个元素的编号。
callConversionTable	包含一个转换表的名称, 当程序在运行时执行下列操作时, 此转换表用来转换数据: <ul style="list-style-type: none"> • 在调用中将自变量传递给远程系统上的程序 • 当通过系统函数 <code>sysLib.startTransaction</code> 调用远程程序时传递自变量 • 访问位于远程位置的文件
errorCode	在发生下列任何一个事件之后接收状态码: <ul style="list-style-type: none"> • 调用 <code>call</code> 语句 (如果该语句位于 <code>try</code> 块中的话) • 对索引文件、MQ 文件、相对文件或串行文件执行 I/O 操作 • 在下列情况下, 调用几乎任何系统函数: <ul style="list-style-type: none"> – 该调用位于 <code>try</code> 块中; 或者 – 程序正在以 VisualAge Generator 兼容性方式运行, 并且 VGVar.handleSysLibraryErrors 设置为 1
formConversionTable	包含当 EGL 生成的 Java 程序执行下列操作时用于双向文本转换的转换表的名称: <ul style="list-style-type: none"> • 显示一个文本或打印表单, 该表单包含一系列希伯莱语或阿拉伯语字符; 或者 • 显示一个文本表单, 该表单从用户那里接受一系列希伯莱语或阿拉伯语字符。
overflowIndicator	当发生算术溢出时设置为 1。通过检查此变量的值, 可以测试溢出条件。
returnCode	包含一个外部返回码, 该返回码是由您的程序设置的, 并且可供操作系统使用。
sessionID	包含特定于 Web 应用程序服务器会话的标识。
sqlca	包含整个 SQL 通信区 (SQLCA)。
sqlcode	包含最近完成的 SQL I/O 操作的返回码。此代码是从 SQL 通信区 (SQLCA) 获取的, 并可以随着关系数据库管理器的不同而有所变化。
sqlState	包含最近完成的 SQL I/O 操作的 SQL 状态值。此代码是从 SQL 通信区 (SQLCA) 获取的, 并可以随着关系数据库管理器的不同而有所变化。
systemType	标识要运行程序的目标系统。

系统变量	描述
terminalID	是从 Java 虚拟机系统属性 <i>user.name</i> 初始化的，并且在不能接收该属性时为空白。
transactionID	如主题 <i>transactionID</i> 中所述。
transferName	允许在运行时指定要转移至的程序或事务的名称。
userID	包含环境中的用户标识（如果有的话）。

相关概念

第 53 页的『引用 EGL 中的变量』

第 52 页的『EGL 中的确定作用域规则和“this”』

相关参考

第 844 页的『EGL 库外部的系统变量』

arrayIndex

系统变量 **SysVar.arrayIndex** 包含一个数字：

- 数组中的第一个与搜索条件相匹配的元素的编号，该搜索条件是带有 **in** 运算符的简单逻辑表达式，如后面的示例所示。
- 零，如果没有任何数组元素与搜索条件相匹配的话。
- 在 **move ... for count** 语句之后，在目标数组中修改的最后一个元素的编号。

可以按照下列任何方式使用 **SysVar.arrayIndex**：

- 作为数组下标，以访问匹配行或数组元素
- 作为赋值语句或 **move** 语句中的源或目标
- 作为 **move...for count** 语句中的计数值
- 作为逻辑表达式中的变量
- 作为 **exit** 语句或 **return** 语句中的自变量

SysVar.arrayIndex 的特征如下所示：

基本类型

BIN

数据长度

4

在转换之后是否始终将值恢复？

仅在非分段文本程序中才如此；有关详细信息，请参阅分段。

示例： 假定记录 *myRecord* 基于以下部件：

```
Record mySerialRecPart
  serialRecord:
    fileName = "myFile"
  end
  10 zipCodeArray CHAR(9)[100];
  10 cityStateArray CHAR(30)[100];
end
```

此外，假定数组是使用邮政编码与“城市和省”的组合初始化的。

以下代码将变量 `currentCityState` 设置为与指定的邮政编码相对应的城市和省:

```
currentZipCode = "27540";
if (currentZipCode in myRecord.zipCodeArray)
    currentCityState = myRecord.cityStateArray[SysVar.arrayIndex];
end
```

在 `if` 语句后面, **SysVar.arrayIndex** 包含第一个包含“27540”值的 `zipCodeArray` 元素的下标。如果在 `zipCodeArray` 中找不到“27540”, 则 **SysVar.arrayIndex** 的值为 0。

相关概念

第 147 页的『文本应用程序中的分段』

相关参考

第 68 页的『数组』

第 486 页的『in 运算符』

第 454 页的『逻辑表达式』

第 844 页的『EGL 库外部的系统变量』

callConversionTable

系统变量 **SysVar.callConversionTable** 包含一个转换表的名称, 当程序在运行时执行下列操作时, 此转换表用来转换数据:

- 在调用中将自变量传递给远程系统上的程序
- 当通过系统函数 `SysLib.startTransaction` 调用远程程序时传递自变量
- 访问位于远程位置的文件

当在基于 EBCDIC 的系统与基于 ASCII 的系统之间或者在使用不同代码页的系统之间移动数据时, 就会发生转换。仅当在生成时使用的链接选项部件在 `callLink` 或 `asynchLink` 元素中指定了 **PROGRAMCONTROLLED** 作为 **conversionTable** 属性的值时, 才有可能进行转换。但是, 如果指定了 **PROGRAMCONTROLLED**, 但 **SysVar.callConversionTable** 是空白的, 则不会发生转换。

特征: **SysVar.callConversionTable** 的特征如下所示:

基本类型

CHAR

数据长度

8

是否跨段保存值?

是

定义注意事项: 应该使用 **SysVar.callConversionTable** 来在程序中切换转换表, 或者在程序中打开或关闭数据转换。

SysVar.callConversionTable 被初始化为空白。要使转换发生, 请确保链接选项部件包含 **PROGRAMCONTROLLED** 值 (如前所述), 并将转换表的名称移动到系统变量中。可以将 **SysVar.callConversionTable** 设置为星号 (*) 以使用缺省本地语言代码的缺省转换表。此设置引用目标系统上的缺省语言环境 (倘若该语言环境映射至其中一种可对 **targetNLS** 构建描述符选项指定的语言的话)。

转换是在发起调用或文件访问的系统上执行的。当定义多级记录结构时，转换是对最低级别的项（没有子结构的项）执行的。

可以按照下列方式使用 **SysVar.callConversionTable**:

- 作为赋值语句或 **move** 语句中的源或目标操作数
- 作为逻辑表达式中的变量
- 作为 **return** 或 **exit** 语句中的自变量

对于 **SysVar.callConversionTable** 与另一个值的比较，仅当完全匹配时测试结果才为 true。例如，如果用小写值来初始化 **SysVar.callConversionTable**，则小写值只与小写值相匹配。

为了进行比较，您对 **SysVar.callConversionTable** 指定的值保持不变。

示例:

```
SysVar.callConversionTable = "ELACNENU";  
// conversion table for US English COBOL generation
```

相关参考

第 426 页的『数据转换』

第 835 页的『startTransaction()』

第 844 页的『EGL 库外部的系统变量』

第 368 页的『targetNLS』

errorCode

系统变量 **SysVar.errorCode** 在下列任何一个事件发生之后接收状态码:

- 调用 call 语句（如果该语句位于 try 块中的话）
- 对索引文件、MQ 文件、相对文件或串行文件执行 I/O 操作
- 在下列情况下，调用几乎任何系统函数:
 - 该调用位于 try 块中；或者
 - 程序正在以 VisualAge Generator 兼容性方式运行，并且 **VGVar.handleSysLibraryErrors** 设置为 1

与给定系统函数相关联的 **SysVar.errorCode** 值在与系统函数相关的主题中进行描述，在当前主题中不进行描述。

可以按照下列方式使用 **SysVar.errorCode**:

- 作为赋值语句或 **move** 语句中的源或目标
- 作为逻辑表达式中的变量
- 在函数调用中，作为与 in、out 或 inOut 参数相关联的自变量

如果调用、I/O 或系统函数调用成功，则 **SysVar.errorCode** 将设置为 0。

SysVar.errorCode 的特征如下所示:

基本类型

CHAR

数据长度

8

在转换之后是否始终将值恢复？

是

要获取包括有关 **SysVar.errorCode** 的详细信息概述，请参阅异常处理。EGL Java 运行时错误代码提供了可能的 **SysVar.errorCode** 值的列表。

示例:

```
if (SysVar.errorCode == "00000008")
  exit program;
end
```

相关参考

第 879 页的『EGL Java 运行时错误代码』

第 86 页的『异常处理』

第 844 页的『EGL 库外部的系统变量』

第 590 页的『try』

第 869 页的『handleSysLibraryErrors』

formConversionTable

系统变量 **SysVar.formConversionTable** 包含当 EGL 生成的 Java 程序执行下列操作时用于双向文本转换的转换表的名称:

- 显示一个文本或打印表单，该表单包含一系列希伯来语或阿拉伯语字符；或者
- 显示一个文本表单，该表单从用户那里接受一系列希伯来语或阿拉伯语字符。

特征: **SysVar.formConversionTable** 的特征如下所示:

基本类型

CHAR

数据长度

8

是否跨段保存值？

是

相关参考

第 429 页的『双向语言文本』

第 426 页的『数据转换』

第 844 页的『EGL 库外部的系统变量』

overflowIndicator

当发生算术溢出时，系统变量 **SysVar.overflowIndicator** 设置为 1。通过检查此变量的值，可以测试溢出条件。

在检测到溢出条件之后，**SysVar.overflowIndicator** 不会自动重置。在执行可能会触发溢出检查的任何计算之前，必须在程序中包括代码以将 **SysVar.overflowIndicator** 重置为 0。

可以按照下列方式使用 **SysVar.overflowIndicator**:

- 作为赋值语句或 **move** 语句中的源或目标（在 **move** 语句的“for count”中也是允许的）
- 作为逻辑表达式中的变量
- 作为 **exit** 语句或 **return** 语句中的自变量

SysVar.overflowIndicator 的特征如下所示:

基本类型

NUM

数据长度

1

在转换之后是否始终将值恢复?

是

示例:

```
SysVar.overflowIndicator = 0;
VGVar.handleOverflow = 2;
a = b;
if (SysVar.overflowIndicator == 1)
    add errorrecord;
end
```

相关参考

第 340 页的『赋值』

第 844 页的『EGL 库外部的系统变量』

第 868 页的『handleOverflow』

returnCode

系统变量 **SysVar.returnCode** 包含一个外部返回码，该返回码是由您的程序设置的，并且可供操作系统使用。将返回码从一个 EGL 程序传递至另一个 EGL 程序是不可能的。例如，非零返回码不会导致 EGL 运行 onException 块。

SysVar.returnCode 的初始值为零，并且该值必须在 -2147483648 到 2147483647 之间（包括这两个数字）。

SysVar.returnCode 仅对主文本程序（它在 J2EE 外部运行）或主批处理程序（它在 J2EE 外部或 J2EE 应用程序客户机中运行）才有意义。在此上下文中，**SysVar.returnCode** 的用途是为调用程序的命令文件或可执行文件提供代码。如果程序由于未受程序控制的错误而结束，则 EGL 运行时忽略 **SysVar.returnCode** 的设置并尝试返回 693 值。

可以按照下列方式使用 **SysVar.returnCode**:

- 作为赋值语句或 **move** 语句中的源或目标（在 **move** 语句的“for count”中也是允许的）
- 作为逻辑表达式中的变量
- 作为 **exit** 或 **return** 语句中的自变量

SysVar.returnCode 的特征如下所示:

基本类型

BIN

数据长度

9

在转换之后是否始终将值恢复？

是

示例:

```
SysVar.returnValue = 6;
```

相关参考

第 844 页的『EGL 库外部的系统变量』

sessionID

在 Web 应用程序中，系统变量 **SysVar.sessionID** 包含特定于 Web 应用程序服务器会话的标识。可以使用 **SysVar.sessionID** 值作为键值来访问在程序之间共享的文件或数据库信息。

在 Web 应用程序外部，下列描述是适用的：

- 系统变量 **SysVar.sessionID** 包含对应于系统的程序的用户标识或终端标识
- 仅仅是为了与 EGL 之前的产品（确切地说，是 CSP 370AD V4R1 之前的 CSP 发行版）兼容才支持以此方式使用 **SysVar.sessionID**。建议您改为使用 **SysVar.userID** 或 **SysVar.terminalID**。

可以按照下列方式使用 **SysVar.sessionID**：

- 作为赋值语句或 **move** 语句中的源
- 作为逻辑表达式中的变量
- 作为 **return** 语句中的自变量

SysVar.sessionID 的特征如下所示：

基本类型

CHAR

数据长度

8（如果值少于 8 个字符，则用空格填充）

在转换之后是否始终将值恢复？

是

SysVar.sessionID 是根据 Java 虚拟机系统属性 *user.name* 初始化的；如果无法检索到该属性，则 **SysVar.sessionID** 是空白的。

示例:

```
myItem = SysVar.sessionID;
```

相关参考

第 844 页的『EGL 库外部的系统变量』

第 860 页的『terminalID』

第 861 页的『userID』

sqlca

系统变量 **SysVar.sqlca** 包含整个 SQL 通信区 (SQLCA)。如后文所述, 在代码访问关系数据库之后, SQLCA 中部分字段的当前值便可供使用。

可以按照下列方式使用 **SysVar.sqlca**:

- 作为赋值语句或 **move** 语句中的源或目标
- 在函数调用中, 作为与 in、out 或 inOut 参数相关联的自变量
- 作为逻辑表达式中的变量
- 作为 **exit** 语句或 **return** 语句中的自变量

为了引用 SQLCA 中的特定字段, 必须将 **SysVar.sqlca** 移至基本记录。该记录必须具有数据库管理系统的 SQLCA 描述中指定的结构。如果将 SQLCA 内容传递至远程程序以便能够正确地将内容转换为远程系统数据格式, 则使用基本记录。

有关 **SysVar.sqlca** 中的可用字段的特定信息, 参阅下列主题:

- VGVVar.sqlerrd
- SysVar.sqlcode
- SysVar.sqlState
- VGVVar.sqlWarn

SysVar.sqlca 的特征如下所示:

基本类型

HEX

数据长度

272 (136 个字节)

在转换之后是否始终将值恢复?

仅在非分段文本程序中才如此; 有关详细信息, 请参阅分段。

示例:

```
myItem = SysVar.sqlca;
```

相关概念

第 147 页的『文本应用程序中的分段』

第 209 页的『SQL 支持』

相关参考

第 844 页的『EGL 库外部的系统变量』

第 858 页的『sqlcode』

第 858 页的『sqlState』

第 870 页的『sqlerrd』

第 872 页的『sqlWarn』

sqlcode

系统变量 **SysVar.sqlcode** 包含最近完成的 SQL I/O 操作的返回码。此代码是从 SQL 通信区 (SQLCA) 获取的, 并可以随着关系数据库管理器的不同而有所变化。

可以按照下列方式使用 **SysVar.sqlcode**:

- 作为赋值语句或 **move** 语句中的源或目标 (在 **move** 语句的“for count”中也是允许的)
- 在函数调用中, 作为与 in、out 或 inOut 参数相关联的自变量
- 作为逻辑表达式中的变量
- 作为 **exit** 语句或 **return** 语句中的自变量

SysVar.sqlcode 的特征如下所示:

基本类型

BIN

数据长度

9

在转换之后是否始终将值恢复?

仅在非分段文本程序中才如此; 有关详细信息, 请参阅分段。

示例:

```
rcitem = SysVar.sqlcode;
```

相关概念

第 147 页的『文本应用程序中的分段』

第 209 页的『SQL 支持』

相关参考

第 844 页的『EGL 库外部的系统变量』

sqlState

系统变量 **SysVar.sqlState** 包含最近完成的 SQL I/O 操作的 SQL 状态值。此代码是从 SQL 通信区 (SQLCA) 获取的, 并可以随着关系数据库管理器的不同而有所变化。

可以按照下列方式使用 **SysVar.sqlState**:

- 作为赋值语句或 **move** 语句中的源或目标
- 在函数调用中, 作为与 in、out 或 inOut 参数相关联的自变量
- 作为逻辑表达式中的变量
- 作为 **exit** 语句或 **return** 语句中的自变量

SysVar.sqlState 的特征如下所示:

基本类型

CHAR

数据长度

5

在转换之后是否始终将值恢复？

仅在非分段文本程序中才如此；有关详细信息，请参阅分段。

示例:

```
rcitem = SysVar.sqlState;
```

相关概念

第 147 页的『文本应用程序中的分段』

第 209 页的『SQL 支持』

相关参考

第 844 页的『EGL 库外部的系统变量』

systemType

系统变量 **SysVar.systemType** 标识正在其中运行程序的目标系统。如果生成的输出是 Java 包装器，则 **SysVar.systemType** 不可用。否则，有效值如下所示:

aix 表示 AIX

debug 表示 EGL 调试器

hp 表示 HP-UX

iseriesj

表示 iSeries

linux 表示 Linux (在基于 Intel 的硬件上)

solaris

表示 Solaris

win 表示 Windows 2000/NT/XP

可以按照下列方式使用 **SysVar.systemType**:

- 作为赋值语句或 **move** 语句中的源
- 作为逻辑表达式中的变量
- 作为 **return** 语句中的自变量

SysVar.systemType 的特征如下所示:

基本类型

CHAR

数据长度

8 (用空格填充)

在转换之后是否始终将值恢复？

是

使用 **SysVar.systemType** 而不是 **VGLib.getVAGSysType**。

定义注意事项: **SysVar.systemType** 的值不影响在生成时验证的代码。例如，即使正在为 Windows 进行生成，也验证以下 **add** 语句:

```
if (sysVar.systemType IS AIX)
  add myRecord;
end
```

为了避免验证完全不会在目标系统上运行的代码，请执行下列其中一项操作：

- 将构建描述符选项 **EliminateSystemDependentCode** 设置为 YES。在当前示例中，如果将该构建描述符选项设置为 YES，则不验证 **add** 语句。但是，您应该了解，仅当逻辑表达式（在此例中，这是 `SysVar.systemType IS AIX`）足够简单从而能够在生成时被求值时，生成器才可以消除对应于系统的代码。
- 另外，将不想验证的语句移至第二个程序；然后，让原始程序有条件地调用新程序：

```
if (SysVar.systemType IS AIX)
  call myAddProgram myRecord;
end
```

示例：

```
if (SysVar.systemType is WIN)
  call myAddProgram myRecord;
end
```

相关参考

第 354 页的『`eliminateSystemDependentCode`』

第 844 页的『EGL 库外部的系统变量』

第 842 页的『`getVAGSysType()`』

terminalID

SysVar.terminalID（与 **SysVar.sessionID** 相似）是根据 Java 虚拟机系统属性 `user.name` 初始化的，并且如果无法检索该属性，则 **SysVar.terminalID** 是空白的。

可以按照下列方式使用 **SysVar.terminalID**：

- 作为赋值语句或 **move** 语句中的源
- 作为逻辑表达式中的变量
- 作为 **return** 语句中的自变量

SysVar.terminalID 的特征如下所示：

基本类型

CHAR

数据长度

8，如果该值的长度小于最大字符数，则用空格对其进行填充

在转换之后是否始终将值恢复？

是

示例：

```
myItem10 = SysVar.terminalID;
```

transactionID

不使用该变量；但如果程序是由 `transfer to program` 格式的 **transfer** 语句调用的，则该变量包含转移程序的名称。

可以按照下列任何方式来使用此变量：

- 作为赋值语句或 **move** 语句中的源或目标

- 作为逻辑表达式中的比较值
- 作为 `return` 语句中的值

SysVar.transactionID 的特征如下所示:

基本类型

CHAR

数据长度

8

在转换之后是否始终将值恢复?

是

相关概念

第 147 页的『文本应用程序中的分段』

相关参考

第 844 页的『EGL 库外部的系统变量』

transferName

系统变量 **SysVar.transferName** 允许在运行时指定要转移至的程序或事务的名称。

可以按照下列任何方式来使用此变量:

- 作为赋值语句或 `move` 语句中的源或目标
- 作为 `transfer` 语句中的程序名或事务名
- 作为逻辑表达式中的比较值
- 作为 `return` 语句中的值

SysVar.transferName 的特征如下所示:

基本类型

CHAR

数据长度

8

在转换之后是否始终将值恢复?

是

相关参考

第 844 页的『EGL 库外部的系统变量』

第 589 页的『`transfer`』

userID

系统变量 **SysVar.userID** 包含环境中的用户标识（如果有的话）。

可以按照下列方式使用 **SysVar.userID**:

- 作为赋值语句或 `move` 语句中的源
- 作为逻辑表达式中的变量
- 作为 `return` 语句中的自变量

SysVar.userID 的特征如下所示:

基本类型

CHAR

数据长度

8 (如果值少于 8 个字符, 则用空格填充)

在转换之后是否始终将值恢复?

是

SysVar.userID 是根据 Java 虚拟机系统属性 *user.name* 初始化的; 如果无法检索到该属性, 则 **SysVar.userID** 是空白的。

示例:

```
myItem = SysVar.userID;
```

VGVar

限定符 **VGVar** 可以放在下表中列出的每个 EGL 系统变量的名称的前面。这些变量主要用于从 VisualAge Generator 迁移的应用程序。

系统变量	描述
currentFormattedGregorianDate	包含长格里历格式的当前系统日期。
currentFormattedJulianDate	包含长儒略历格式的当前系统日期。
currentFormattedTime	包含 HH:mm:ss 格式的当前系统时间。
currentGregorianDate	包含 8 位格里历格式 (yyyyMMdd) 的当前系统日期。
currentJulianDate	包含 7 位儒略历格式 (yyyyDDD) 的当前系统日期。尽量避免使用此变量, 此变量用于支持从 VisualAge Generator 至 EGL 的代码迁移。
currentShortGregorianDate	包含 6 位格里历格式 (yyMMdd) 的当前系统日期。尽量避免使用此变量, 此变量用于支持从 VisualAge Generator 至 EGL 的代码迁移。
currentShortJulianDate	包含 5 位儒略历格式 (yyDDD) 的当前系统日期。尽量避免使用此变量, 此变量用于支持从 VisualAge Generator 至 EGL 的代码迁移。
handleHardIOErrors	控制在 try 块中的 I/O 操作发生硬错误后程序是否继续运行。
handleOverflow	控制算术溢出后的错误处理。
handleSysLibraryErrors	指定系统变量 SysVar.errorCode 的值是否受系统函数调用的影响。
mqConditionCode	包含在对 MQ 记录执行 add 或 scan I/O 操作之后进行的 MQSeries API 调用的完成代码。
sqlerrd	包含 6 个元素的数组, 其中每个元素都包含从上一个 SQL I/O 选项返回的相应 SQL 通信区 (SQLCA) 值。
sqlerrmc	包含与 SysVar.sqlcode 中的返回码相关联的错误消息的替换变量。

系统变量	描述
sqlIsolationLevel	指示一个数据库事务相对于另一数据库事务的独立性级别。
sqlWarn	包含 11 个元素的数组，其中每个元素包含在上一个 SQL I/O 操作的 SQL 通信区 (SQLCA) 中返回的警告字节，并且下标比 SQL SQLCA 描述中的警告编号大 1。

相关概念

第 53 页的『引用 EGL 中的变量』

第 52 页的『EGL 中的确定作用域规则和“this”』

相关参考

第 844 页的『EGL 库外部的系统变量』

currentFormattedGregorianDate

系统变量 **VGVar.currentFormattedGregorianDate** 包含长格里历格式的当前系统日期。程序每次引用此系统变量时，值都会自动地更新。

格式在以下 Java 运行时属性中：

```
vgj.datemask.gregorian.long.NLS
```

NLS

Java 运行时属性 **vgj.nls.code** 中指定的 NLS（本地语言支持）代码。此代码是对 targetNLS 构建描述符选项列示的那些代码中的一个。不支持大写英语（代码 ENP）

有关 **vgj.nls.code** 的详细信息，请参阅 *Java 运行时属性*（[详细信息](#)）。

vgj.datemask.gregorian.long.NLS 中指定的格式包含 dd（表示数字格式的天）、MM（表示数字格式的月份）和 yyyy（表示数字格式的年），并将除 d、M、y 或数字之外的字符用作分隔符。可以在 **dateMask** 构建描述符选项中指定格式，而缺省格式是特定于语言环境的。

可以将 **VGVar.currentFormattedGregorianDate** 用作赋值或 **move** 语句中的源，也可以用作 **return** 或 **exit** 语句中的自变量。

确保此长格里历日期格式与对 SQL 数据库管理器指定的格式相同。使两种格式匹配将使 **VGVar.currentFormattedGregorianDate** 能够生成具有数据库管理器所期望的格式的日期。

VGVar.currentFormattedGregorianDate 的特征如下：

基本类型

CHAR

数据长度

10

跨段保存值

否

示例:

```
myDate = VGVar.currentFormattedGregorianCalendarDate;
```

相关概念

第 267 页的『构建描述符部件』

第 315 页的『Java 运行时属性』

相关任务

第 275 页的『编辑构建描述符中的 Java 运行时属性』

相关参考

第 723 页的『EGL 库 DateTimeLib』

第 493 页的『Java 运行时属性（详细信息）』

第 844 页的『EGL 库外部的系统变量』

第 368 页的『targetNLS』

currentFormattedJulianDate

系统变量 **VGVar.currentFormattedJulianDate** 包含长儒略历格式的当前系统日期。程序每次引用此系统变量时，值都会自动地更新

格式在以下 Java 运行时属性中：

```
vgj.datemask.julian.long.NLS
```

NLS

Java 运行时属性 **vgj.nls.code** 中指定的 NLS（本地语言支持）代码。此代码是对 targetNLS 构建描述符选项列示的那些代码中的一个。不支持大写英语（代码 ENP）

有关 **vgj.nls.code** 的详细信息，请参阅 *Java 运行时属性（详细信息）*。

vgj.datemask.julian.long.NLS 中指定的格式包含 DDD（表示数字格式的天）和 yyyy（表示数字格式的年），并将除 D、y 或数字之外的字符用作分隔符。可以在 **dateMask** 构建描述符选项中指定格式，而缺省格式是特定于语言环境的。

可以将 **VGVar.currentFormattedJulianDate** 用作赋值或 **move** 语句中的源，也可以用作 **return** 或 **exit** 语句中的自变量。

VGVar.currentFormattedJulianDate 的特征如下：

基本类型

CHAR

数据长度

8

跨段保存值

否

不支持大写英语（NLS 代码 ENP）。

示例：

```
myDate = VGVar.currentFormattedJulianDate;
```

相关概念

第 267 页的『构建描述符部件』

第 315 页的『Java 运行时属性』

相关任务

第 275 页的『编辑构建描述符中的 Java 运行时属性』

相关参考

第 723 页的『EGL 库 DateTimeLib』

第 493 页的『Java 运行时属性（详细信息）』

第 844 页的『EGL 库外部的系统变量』

第 368 页的『targetNLS』

currentFormattedTime

系统变量 **VGVar.currentFormattedTime** 包含 HH:mm:ss 格式的当前系统时间。程序每次引用此值时，此值都会自动地更新。

可以按照下列方式使用 **VGVar.currentFormattedTime**:

- 作为赋值语句或 **move** 语句中的源
- 作为 **exit** 语句或 **return** 语句中的自变量

VGVar.currentFormattedTime 的特征如下:

基本类型

CHAR

数据长度

8

跨段保存值

否

示例:

```
timeField = VGVar.currentFormattedTime;
```

相关参考

第 723 页的『EGL 库 DateTimeLib』

第 844 页的『EGL 库外部的系统变量』

currentGregorianDate

系统变量 **VGVar.currentGregorianDate** 包含 8 位格里历格式 (yyyyMMdd) 的当前系统日期。

在每次引用 **VGVar.currentGregorianDate** 值之前，它都会自动地更新。此值是数字，并且不包含分隔符。

可以将 **VGVar.currentGregorianDate** 用作赋值或 **move** 语句中的源，也可以用作 **return** 或 **exit** 语句中的自变量。

VGVar.currentGregorianDate 的特征如下:

基本类型

DATE

数据长度

8

跨段保存值

否

示例:

```
myDate = VGVar.currentGregorianDate
```

相关参考

第 723 页的『EGL 库 DateTimeLib』

第 844 页的『EGL 库外部的系统变量』

currentJulianDate

系统变量 **VGVar.currentJulianDate** 包含 7 位儒略历格式 (yyyyDDD) 的当前系统日期。尽量避免使用此变量，此变量用于支持从 VisualAge Generator 至 EGL 的代码迁移。

此值为数字值，没有任何分隔符，每次引用前，此值都会自动地更新。

可以将 **VGVar.currentJulianDate** 用作赋值或 **move** 语句中的源，也可以用作 **return** 或 **exit** 语句中的自变量。

VGVar.currentJulianDate 的特征如下:

基本类型

NUM

数据长度

7

跨段保存值

否

示例:

```
myDay = VGVar.currentJulianDate;
```

相关参考

第 723 页的『EGL 库 DateTimeLib』

第 844 页的『EGL 库外部的系统变量』

currentShortGregorianDate

系统变量 **VGVar.currentShortGregorianDate** 包含 6 位格里历格式 (yyMMdd) 的当前系统日期。尽量避免使用此变量，此变量用于支持从 VisualAge Generator 至 EGL 的代码迁移。

每当程序引用 **VGVar.currentShortGregorianDate** 值时，都会自动更新该值。返回的值是数字，并且不包含分隔符。

可以将 **VGVar.currentShortGregorianDate** 用作赋值或 **move** 语句中的源，也可以用作 **return** 或 **exit** 语句中的自变量。

VGVar.currentShortGregorianDate 的特征如下:

基本类型

NUM

数据长度

6

跨段保存值

否

示例:

```
myDay = VGVar.currentShortGregorianDate;
```

相关参考

第 723 页的『EGL 库 DateTimeLib』

第 844 页的『EGL 库外部的系统变量』

currentShortJulianDate

系统变量 **VGVar.currentShortJulianDate** 包含 5 位儒略历格式 (yyDDD) 的当前系统日期。尽量避免使用此变量，此变量用于支持从 VisualAge Generator 至 EGL 的代码迁移。

此值为数字值，没有任何分隔符，程序每次引用此值时，此值都会自动地更新。

可以将 **VGVar.currentShortJulianDate** 用作赋值或 **move** 语句中的源，也可以用作 **return** 或 **exit** 语句中的自变量。

VGVar.currentShortJulianDate 的特征如下:

基本类型

NUM

数据长度

5

跨段保存值

否

示例:

```
myDay = VGVar.currentShortJulianDate;
```

相关参考

第 723 页的『EGL 库 DateTimeLib』

第 844 页的『EGL 库外部的系统变量』

handleHardIOErrors

系统变量 **VGVar.handleHardIOErrors** 控制在 **try** 块中的 I/O 操作发生硬错误之后程序是否继续运行。缺省值为 1，除非您将程序属性 **handleHardIOErrors** 设置为 *no*，这会将变量设置为 0。（该属性对其它可生成逻辑部件也是可用的。）有关背景知识信息，请参阅异常处理。

可以按照下列任何方式使用 **VGVar.handleHardIOErrors**:

- 作为赋值语句或 **move** 语句的源或目标（在 **move** 语句的“for count”中也是允许的）
- 作为在 **case** 语句、**if** 语句或 **while** 语句中使用的逻辑表达式中的变量
- 作为 **return** 语句或 **exit** 语句中的自变量

VGVar.handleHardIOErrors 的特征如下:

基本类型

NUM

数据长度

1

在转换之后是否始终将值恢复?

是

示例

```
VGVar.handleHardIOErrors = 1;
```

相关参考

第 86 页的『异常处理』

第 844 页的『EGL 库外部的系统变量』

handleOverflow

系统变量 **VGVar.handleOverflow** 控制算术溢出之后的错误处理。检测两种类型的溢出条件:

- 当算术运算或数字项赋值的结果由于项的长度不足而导致有效值（不是小数位）丢失时，发生用户变量溢出。
- 当算术运算的结果大于 18 位时，发生最大值溢出。

可以将 **VGVar.handleOverflow** 设置为下列其中一个值。（缺省设置为 0。）

值	对用户溢出的影响	对最大值溢出的影响
0	程序将系统变量 SysVar.overflowIndicator 设置为 1 并继续	程序结束，并发出错误消息
1	程序结束，并发出错误消息	程序结束，并发出错误消息
2	程序将系统变量 SysVar.overflowIndicator 设置为 1 并继续	程序将系统变量 SysVar.overflowIndicator 设置为 1 并继续

可以按照下列方式使用 **VGVar.handleOverflow**:

- 作为赋值语句或 **move** 语句中的源或目标（在 **move** 语句的“for count”中也是允许的）
- 作为逻辑表达式中的变量
- 作为 **exit** 语句或 **return** 语句中的自变量

VGVar.handleOverflow 的特征如下:

基本类型

NUM

数据长度

1

在转换之后是否始终将值恢复？

是

示例:

```
VGVar.handleOverflow = 2;
```

相关参考

第 340 页的『赋值』

第 844 页的『EGL 库外部的系统变量』

第 854 页的『overflowIndicator』

handleSysLibraryErrors

系统变量 **VGVar.handleSysLibraryErrors** 指定系统变量 **SysVar.errorCode** 的值是否受系统函数调用的影响。但是，仅当 VisualAge Generator 兼容性起作用时，**VGVar.handleSysLibraryErrors** 才可用，如与 *VisualAge Generator* 的兼容性中所述。

有关详细信息和限制，请参阅异常处理。

可以按照下列方式使用 **VGVar.handleSysLibraryErrors**:

- 作为赋值语句或 **move** 语句中的源或目标
- 作为逻辑表达式中的变量
- 作为 **exit** 语句或 **return** 语句中的自变量

VGVar.handleSysLibraryErrors 的特征如下:

基本类型

NUM

数据长度

1

在转换之后是否始终将值恢复？

仅在非分段文本程序中才如此；有关详细信息，请参阅分段。

示例:

```
VGVar.handleSysLibraryErrors = 1;
```

相关概念

第 400 页的『与 VisualAge Generator 的兼容性』

第 147 页的『文本应用程序中的分段』

相关参考

第 86 页的『异常处理』

第 844 页的『EGL 库外部的系统变量』

第 853 页的『errorCode』

mqConditionCode

系统变量 **VGVar.mqConditionCode** 包含在对 MQ 记录执行 **add** 或 **scan** I/O 操作之后进行的 MQSeries API 调用的完成代码。有效值及其相关含义如下所示:

00 正常

01 警告

02 失败

可以按照下列方式使用 **VGVar.mqConditionCode**:

- 作为赋值语句或 **move** 语句中的源或目标（在 **move** 语句的“for count”中也是允许的）
- 作为逻辑表达式中的变量
- 作为 **exit** 语句或 **return** 语句中的自变量

VGVar.mqConditionCode 的特征如下:

基本类型

NUM

数据长度

2

在转换之后是否始终将值恢复?

是

示例:

```
add MQRecord;
if (VGVar.mqConditionCode == 0)
    // continue
else
    exit program;
end
```

相关概念

第 242 页的『MQSeries 支持』

相关参考

第 86 页的『异常处理』

第 844 页的『EGL 库外部的系统变量』

sqlerrd

系统数组 **VGVar.sqlerrd** 是包含 6 个元素的数组，其中每个元素都包含从上一个 SQL I/O 选项返回的相应 SQL 通信区 (SQLCA) 值。例如，**VGVar.sqlerrd[3]** 中的值是第 3 个值，它指示对某些 SQL 请求处理的行数。

在 **VGVar.sqlerrd** 的元素当中，对于 Java 代码或者在调试期间，数据库管理系统仅刷新 **VGVar.sqlerrd[3]**。

可以按照下列方式使用 **VGVar.sqlerrd** 元素:

- 作为赋值语句或 **move** 语句中的源或目标
- 作为 **move** 语句的 **for count** 子句中的值
- 在函数调用中，作为与 **in**、**out** 或 **inOut** 参数相关联的自变量
- 作为逻辑表达式中的变量
- 作为 **exit** 语句或 **return** 语句中的自变量

VGVar.sqlerrd 数组中每个元素的特征如下所示:

基本类型

BIN

数据长度

9

在转换之后是否始终将值恢复？

仅在非分段文本程序中才如此；有关详细信息，请参阅分段。

示例:

```
myItem = VGVar.sqlerrd[3];
```

相关概念

第 147 页的『文本应用程序中的分段』

第 209 页的『SQL 支持』

相关参考

第 844 页的『EGL 库外部的系统变量』

sqlerrmc

系统变量 **VGVar.sqlerrmc** 包含与 **SysVar.sqlcode** 中的返回码相关联的错误消息。**VGVar.sqlerrmc** 是从 SQL 通信区 (SQLCA) 获取的，并可以随着关系数据库管理器的不同而有所变化。

VGVar.sqlerrmc 对 JDBC 环境无意义。

可以按照下列方式使用 **VGVar.sqlerrmc**:

- 作为赋值语句或 **move** 语句中的源或目标
- 作为逻辑表达式中的变量
- 在函数调用中，作为与 **in**、**out** 或 **inOut** 参数相关联的自变量
- 作为 **exit** 语句或 **return** 语句中的自变量

VGVar.sqlerrmc 的特征如下

基本类型

CHAR

数据长度

70

在转换之后是否始终将值恢复？

仅在非分段文本程序中才如此；有关详细信息，请参阅分段。

示例:

```
myItem = VGVar.sqlerrmc;
```

相关概念

第 147 页的『文本应用程序中的分段』

第 209 页的『SQL 支持』

相关参考

第 857 页的『sqlca』

第 844 页的『EGL 库外部的系统变量』

sqlIsolationLevel

系统变量 **VGVar.sqlIsolationLevel** 指示一个数据库事务相对于另一数据库事务的独立性级别。

有关隔离级别以及短语可重复读和可序列化事务的概述，请参阅 Sun 公司提供的 JDBC 文档。

VGVar.sqlIsolationLevel 仅用于从 VisualAge Generator 和 EGL 5.0 迁移的程序。如果（在开发时）选择了 EGL 首选项 **VisualAge Generator 兼容性** 或者（在生成时）将构建描述符选项 **VAGCompatibility** 设置为 *yes*，则支持此函数。

对于新的开发工作，在 **SysLib.connect** 语句中设置 SQL 隔离级别。

VGVar.sqlIsolationLevel 的下列值用于提高严密性：

0（缺省值）

可重复读

1 可序列化事务

可以按照下列任何方式来使用此变量：

- 作为赋值语句或 `move` 语句中的源或目标
- 在函数调用中，作为与 `in`、`out` 或 `inOut` 参数相关联的自变量
- 作为逻辑表达式中的比较值
- 作为 `return` 语句中的值

SysVar.transactionID 的特征如下所示：

基本类型

NUM

数据长度

1

在转换之后是否始终将值恢复？

是

相关参考

第 820 页的『connect()』

第 844 页的『EGL 库外部的系统变量』

sqlWarn

系统数组 **VGVar.sqlWarn** 是包含 11 个元素的数组，其中每个元素都包含在上一个 SQL I/O 操作的 SQL 通信区（SQLCA）中返回的警告字节，并且下标比 SQL SQLCA 描述中的警告编号大 1。例如，系统变量 **VGVar.sqlWarn[2]** 指的是 SQLWARN1，后者指示在 I/O 操作中是否截断了项中的字符。

在 `VGVar.sqlWarn` 的元素当中，对于 Java 代码或者在调试期间，数据库管理系统仅刷新系统变量 `VGVar.sqlWarn[2]`。

可以按照下列方式使用 `VGVar.sqlWarn`：

- 作为赋值语句或 `move` 语句中的源或目标
- 作为 `move` 语句的 `for count` 子句中的值
- 作为逻辑表达式中的变量
- 在函数调用中，作为与 `in`、`out` 或 `inOut` 参数相关联的自变量
- 作为 `exit` 语句或 `return` 语句中的自变量

`VGVar.sqlWarn` 数组中每个元素的特征如下所示：

基本类型

CHAR

数据长度

1

在转换之后是否始终将值恢复？

仅在非分段文本程序中才如此；有关详细信息，请参阅分段。

定义注意事项： 如果上一项 SQL I/O 操作由于程序主变量空间不足而导致数据库管理器将字符数据项截断，则 `VGVar.sqlWarn[2]` 包含 `W`。可以使用逻辑表达式来测试特定主变量中的值是否已被截断。有关详细信息，请参阅逻辑表达式中 `trunc` 的参考。

当主变量是数字时，不提供截断警告。数字的小数部分将被截断，并且不给出指示。

示例： 在以下示例中，`my-char-field` 是刚刚处理的 SQL 行记录中的一个字段，`lost-data` 是一个函数，该函数设置一条错误消息，该消息指示 `my-char-field` 的信息已被截断。

```
if (VGVar.sqlWarn[2] == 'W')
  if (my-char-field is trunc)
    lost-data();
  end
end
```

相关概念

第 147 页的『文本应用程序中的分段』

第 209 页的『SQL 支持』

相关参考

第 454 页的『逻辑表达式』

第 844 页的『EGL 库外部的系统变量』

transferToTransaction 元素

链接选项部件的 `transferToTransaction` 元素指定生成的程序如何将控制权转移至事务以及结束处理。此元素包含 `toPgm` 属性，并可以包含下列属性：

- `alias`，如果代码正在转移至一个程序，而该程序的运行时名称与相关程序部件的名称不同，则此属性是必需的
- `externallyDefined`，如果代码正在转移至一个程序，而该程序不是使用 EGL 或 VisualAge Generator 生成的，则此属性是必需的

当目标程序是通过 VisualAge Generator 或（当不存在别名时）EGL 生成时，就不需要指定 **transferToTransaction** 元素。

相关概念

第 282 页的『链接选项部件』

相关任务

第 284 页的『将链接选项部件添加至 EGL 构建文件』

第 287 页的『编辑与转移相关的链接选项部件元素』

相关参考

『与转移相关的链接元素中的 **alias**』

『**transferToTransaction** 元素中的 **externallyDefined**』

与转移相关的链接元素中的 **alias**

在链接选项部件的与转移相关的元素中，属性 **alias** 指定属性 **toPgm** 中标识的程序的运行时名称。

此属性的值必须与声明正在转移至的程序时指定的别名（如果有的话）相匹配。如果在声明该程序时未指定别名，则将属性 **alias** 设置为程序部件的名称或者根本不设置该属性。

相关概念

第 282 页的『链接选项部件』

相关任务

第 284 页的『将链接选项部件添加至 EGL 构建文件』

第 287 页的『编辑与转移相关的链接选项部件元素』

相关参考

第 873 页的『**transferToTransaction** 元素』

transferToTransaction 元素中的 **externallyDefined**

链接选项部件的 **transferToTransaction** 元素的 **externallyDefined** 属性表示是否正在转移至由除 EGL 或 VisualAge Generator 以外的软件生成的程序。有效值为 **no**（缺省值）和 **yes**。

如果指定 **yes**，则 XCTL 实现所有 COBOL 目标系统中的 **transfer** 语句。

如果程序属性 **VAGCompatibility** 设置为 **yes**，则可以在 **transfer** 语句中指定 **externallyDefined**，如与 *VisualAge Generator* 的兼容性中所述。建议改为在 **transferToTransaction** 元素中指定该值，但在任何一个位置中指定值都会使该值生效。

相关概念

第 400 页的『与 VisualAge Generator 的兼容性』

相关任务

第 284 页的『将链接选项部件添加至 EGL 构建文件』

第 287 页的『编辑与转移相关的链接选项部件元素』

使用声明

本节先描述使用声明，然后提供有关如何编写该声明的详细信息：

- 『在程序部件或库部件中』
- 第 877 页的『在 formGroup 部件中』
- 第 878 页的『在 pageHandler 部件中』

背景知识

使用声明允许您方便地引用独立生成的部件中的数据区和函数。例如，程序可以发出使用声明来简化对数据表、库或表单组的引用，但仅当那些部件对程序部件可视时才如此。有关可视性的详细信息，请参阅[对部件的引用](#)。

在大多数情况下，无论使用声明是否生效，都可以从一个部件中引用另一部件中的数据区和函数。例如，如果您正在编写程序，并且没有名为 myLib 的库部件的使用声明，则可以按照以下方式访问名为 myVar 的库变量：

```
myLib.myVar
```

但是，如果在使用声明中包括了库名，则可以按照如下方式引用该变量：

```
myVar
```

仅当符号 myVar 对于程序的每个全局变量和结构项来说都是唯一的时，前面的短格式引用才有效。（如果符号不是唯一的，则会发生错误。）并且，仅当局部变量或参数不同时，符号 myVar 指的才是库中的项。（局部数据区优先于同名的程序全局数据区。）

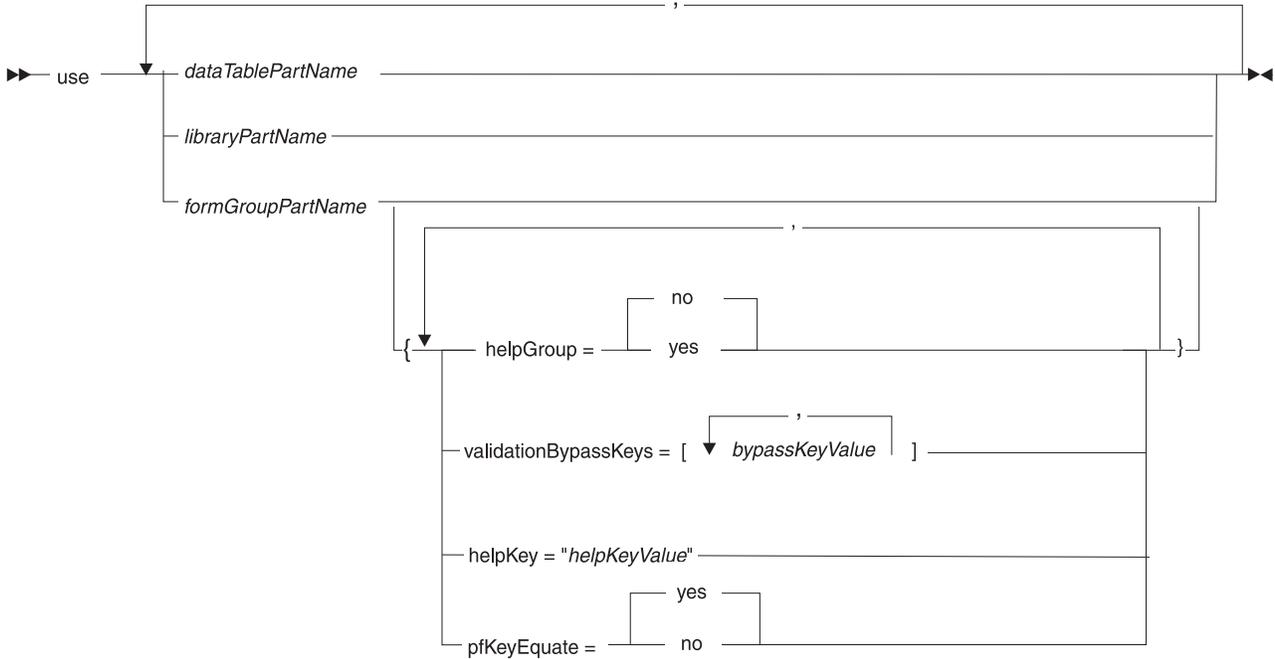
在下列情况下，使用声明是必需的：

- 使用给定 formGroup 部件中的任何表单的程序或库必须要有该 formGroup 部件的使用声明
- formGroup 部件必须要有程序或库所必需的但未嵌入在 formGroup 部件中的表单的使用声明
- 如果在 EGL 源文件的顶部（而不是完全在容器内部，即程序、PageHandler 或库）声明了一个函数，仅当出现下列情况时，该函数才能调用库函数：
 - 容器包括引用库的 use 语句
 - 在调用函数中，属性 **containerContextDependent** 设置为 yes

可以通过包名和 / 或库名来对使用声明中指定的每个名称进行限定。

在程序部件或库部件中

程序或库中的每个使用声明都必须位于任何函数外部。声明的语法如下所示：



dataTablePartName

对程序或库可视的 `dataTable` 部件的名称。

对于在程序属性 `msgTablePrefix` 中引用的 `dataTable` 部件来说，使用声明中的引用不是必需的。

不能在使用声明中覆盖 `dataTable` 部件的属性。

有关 `dataTable` 部件的概述，请参阅 *DataTable* 部件。

libraryPartName

对程序或库可视的库部件的名称。

不能在使用声明中覆盖库部件的属性。

有关库部件的概述，请参阅类型为 *basicLibrary* 的库部件和类型为 *nativeLibrary* 的库部件。

formGroupPartName

对程序或库可视的 `formGroup` 部件的名称。有关表单组的概述，请参阅 *FormGroup* 部件。

使用给定 `formGroup` 部件中的任何表单的程序必须要有该 `formGroup` 部件的使用声明。

表单级别的属性不会被覆盖。例如，如果在表单中指定属性，如 **validationBypassKeys**，则表单中的值在运行时起作用。然而，如果在表单中未指定表单级别的属性，则情况如下所示：

- EGL 运行时使用程序的使用声明中的值
- 如果未在程序的使用声明中指定任何值，则 EGL 运行时使用表单组中的值（如果有的话）

下列属性允许您在特定程序访问表单组时更改行为。

helpGroup = no, helpGroup = yes

指定是否将 formGroup 部件用作帮助组。缺省值为 *no*。

validationBypassKeys = [bypassKeyValue]

标识一个用户击键，该击键导致 EGL 运行时跳过输入字段验证。此属性对于保留用于快速结束程序的击键而言很有用。每个 *bypassKeyValue* 选项如下所示：

pf*n*

F 或 PF 键的名称，包括介于 1 与 24 之间（包括 1 和 24）的数字。

注：PC 键盘上的功能键通常是 *f* 键，如 f1，但 EGL 使用 IBM *pf* 术语，所以 f1（例如）被称为 pf1。

如果要指定多个键，则用逗号来隔开它们。

helpKey = "helpKeyValue"

标识一个用户击键，该击键使 EGL 运行时向用户显示帮助表单。*helpKeyValue* 选项如下所示：

pf*n*

F 或 PF 键的名称，包括介于 1 与 24 之间（包括 1 和 24）的数字。

注：PC 键盘上的功能键通常是 *f* 键，如 f1，但 EGL 使用 IBM *pf* 术语，所以 f1（例如）被称为 pf1。

pfKeyEquate = yes, pfKeyEquate = no

指定当用户按下大编号功能键（PF13 至 PF24）时注册的击键是否与用户按下编号小于 12 的功能键时注册的击键相同。缺省值为 *yes*。有关详细信息，请参阅 *pfKeyEquate*。

在 formGroup 部件中

在 formGroup 部件中，使用声明引用在表单组外部指定的表单。此类声明允许多个表单组共享同一个表单。

formGroup 部件中的使用声明语法如下所示：



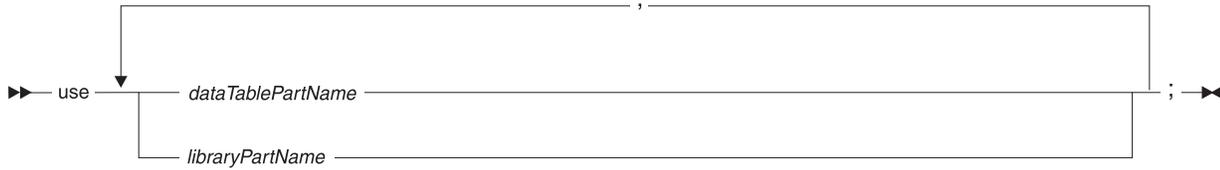
formPartName

对表单组可视的表单部件的名称。有关表单的概述，请参阅 *表单部件*。

不能在 formGroup 部件的使用声明中覆盖表单部件的属性。

在 pageHandler 部件中

pageHandler 部件中的每个使用声明都必须位于任何函数外部。声明的语法如下所示:



dataTablePartName

对 pageHandler 部件可视的 dataTable 部件的名称。

不能在使用声明中覆盖 dataTable 部件的属性。

有关 dataTable 部件的概述, 请参阅 *DataTable 部件*。

libraryPartName

对 pageHandler 部件可视的库部件的名称。

不能在使用声明中覆盖库部件的属性。

有关库部件的概述, 请参阅 *库部件*。

相关概念

第 134 页的『DataTable』

第 141 页的『FormGroup 部件』

第 142 页的『表单部件』

第 131 页的『类型为 basicLibrary 的库部件』

第 131 页的『类型为 basicLibrary 的库部件』

第 20 页的『对部件的引用』

相关参考

第 625 页的『pfKeyEquate』

EGL Java 运行时错误代码

在 Java 运行时发生错误时，EGL 会将错误代码放在系统变量 `sysVar.errorCode` 中并且在大多数情况下会提供一条消息，它的标识与错误代码的标识相同。可显示定制消息以代替 EGL 消息；有关详细信息，请参阅 *EGL Java 运行时的消息定制*。

错误情况如下所示：

- 在远程调用、EJB 调用、落实或回滚期间发生故障。在那些情况下，消息标识以 CSO 开头。
- 在 Web 应用程序中发生错误。在那些情况的部分情况下，消息标识以 EGL 开头。
- 在本地调用期间、访问文件或数据库期间或在执行下列其中一个系统函数期间发生错误：
 - 数学函数
 - 字符串函数
 - `sysLib.convert`

在那些情况下，消息标识以 VGJ 开头。

- 在 Java 访问函数中发生错误。在此情况下，错误代码只包括数字，不会显示任何消息。

下表显示了 Java 访问函数指定的错误代码。其它错误代码显示在下列各节中。

sysVar.errorCode 中的值	描述
00001000	被调用方法抛出了异常，或者由于进行类初始化而抛出了异常。
00001001	对象为 null，或者指定的标识不在对象空间中。
00001002	具有指定名称的公用方法、字段或类不存在或者无法被装入。
00001003	EGL 基本类型与 Java 中期望的类型不匹配。
00001004	方法返回 null，该方法未返回值或字段的值为 null。
00001005	返回的值与返回项的类型不匹配。
00001006	未能装入强制类型转换为 null 的自变量的类。
00001007	在尝试获取有关某个方法或字段的信息时抛出了 SecurityException 或 IllegalAccessException，或者尝试设置声明为 final 的字段的价值。
00001008	不能调用构造函数；类名引用接口或抽象类。
00001009	必须指定标识而不是类名；方法或字段不是静态的。

相关参考

第 490 页的『I/O 错误值』

第 602 页的『EGL Java 运行时的消息定制』

第 853 页的『errorCode』

EGL Java 运行时错误代码 CSO7000E

CSO7000E: 在链接属性文件 %2 中找不到指定的被调用程序 %1 的条目。

说明

在下列情况下将产生此消息:

- 当生成调用程序时, 在链接选项部件中以及被调用程序的 **callLink** 元素中将属性 **remoteBind** 设置为 **RUNTIME**; 并且
- 在运行时在链接属性文件中找不到指定的被调用程序的条目。这可能是由于下列其中一种原因造成的:
 - 找不到链接属性文件。
 - 找到了文件, 但是在该文件中不存在被调用程序的条目。
 - 指定了不正确的链接属性文件。

用户响应

执行下列操作:

- 如果正在从 Java 包装器中调用程序, 则必须将链接属性文件命名为 *link.properties*, 其中 *link* 是在生成时使用的链接选项部件的名称。确保该文件存在、具有被调用程序的条目并且位于在 **CLASSPATH** 变量中指定的目录或归档中。
- 如果是从正在 J2EE 环境中运行的程序中调用程序, 则可以通过部署描述符中的 *csolinkageOptions.link* 环境变量来标识链接属性文件, 其中 *link* 是在生成时使用的链接选项部件的名称。如果未设置该环境变量, 则必须将链接属性文件命名为 *link.properties*, 其中 *link* 是在生成时使用的链接选项部件的名称。确保该文件存在、具有被调用程序的条目并且位于在 **CLASSPATH** 中指定的目录或归档中。
- 如果正在从不是在 J2EE 环境中运行的程序中调用该程序, 则情况如下所示:
 - 可通过 *csolinkageOptions.link* 属性来标识链接属性文件, 其中 *link* 是在生成时使用的链接选项部件的名称。如果未设置该属性, 则可以将链接属性文件命名为 *link.properties*, 其中 *link* 是在生成时使用的链接选项部件的名称。在这两种情况下, 确保该文件存在、具有被调用程序的条目并且位于在 **CLASSPATH** 中指定的目录或归档中。
 - 如果找不到链接属性文件, 则链接属性必须位于程序属性文件中; 在这种情况下, 应确保程序属性文件包含被调用程序的条目, 并且程序属性文件位于在 **CLASSPATH** 中指定的目录或归档中。

有关其它详细信息, 请参阅有关 **callLink** 元素、有关 Java 运行时属性以及有关设置环境的 EGL 帮助页面。

如果问题仍然存在, 则执行以下操作:

1. 记录消息号和消息文本。

注: 错误消息包括下列重要信息:

- 错误的发生位置
- 内部错误的类型

2. 记录发生此消息的情况。

3. 有关如何向“IBM 支持中心”报告可能存在的缺陷的更多指示信息，参阅 *EGL Installation Guide*。

EGL Java 运行时错误代码 CSO7015E

CSO7015E: 无法打开链接属性文件 %1。

说明

由于锁定了或者找不到链接属性文件，所以打不开该文件。

用户响应

确保链接属性文件没有被另一个进程锁定，并且该文件驻留在 CLASSPATH 中指定的目录或归档中。

EGL Java 运行时错误代码 CSO7016E

CSO7016E: 未能读取属性文件 `csoidpwd.properties`。错误: %1

说明

已找到该文件，但是读取它时发生错误。

用户响应

使用此消息的错误部分来诊断和更正问题。

EGL Java 运行时错误代码 CSO7020E

CSO7020E: 转换表 %1 无效。

说明

用来处理双向文本的转换表无效或者未能被装入。

用户响应

转换表必须驻留在 CLASSPATH 中指定的目录或归档中。有关开发转换表的详细信息，请参阅有关双向文本的帮助页面。

EGL Java 运行时错误代码 CSO7021E

CSO7021E: 转换表 %1 中的客户机文本属性标记 %2 无效。

说明

转换表文件无效。

用户响应

更正该文件并再次运行程序。

EGL Java 运行时错误代码 CSO7022E

CSO7022E: 转换表 %1 中的服务器文本属性标记 %2 无效。

说明

转换表文件无效。

用户响应

更正该文件并再次运行程序。

EGL Java 运行时错误代码 CSO7023E

CSO7023E: 转换表 %1 中的阿拉伯语选项标记 %2 的值 %3 无效。

说明

转换表文件无效。

用户响应

更正该文件并再次运行程序。

EGL Java 运行时错误代码 CSO7024E

CSO7024E: 转换表 %1 中的“断字”选项标记 %2 的值 %3 无效。

说明

转换表文件无效。

用户响应

更正该文件并再次运行程序。

EGL Java 运行时错误代码 CSO7026E

CSO7026E: 转换表 %1 中的“来回”选项标记 %2 的值 %3 无效。

说明

转换表文件无效。

用户响应

更正该文件并再次运行程序。

EGL Java 运行时错误代码 CSO7045E

CSO7045E: 获取共享库 %2 中的入口点 %1 的地址时出错。RC = %3。

说明

在获取共享库中的入口点的地址时遇到错误。

用户响应

确保引用的共享库就是要装入的正确共享库。如果是这样的话，确保正确地构建了该共享库。

EGL Java 运行时错误代码 CSO7050E

CSO7050E: 在远程程序 %1 中，在日期 %2 时间 %3 时发生了错误。

说明

在被调用程序中发生了错误，程序已停止运行。

用户响应

使用此消息中的日期和时间戳记来将消息与记录在远程位置的诊断消息相关联。检查那些诊断消息以获取更多详细信息。

EGL Java 运行时错误代码 CSO7060E

CSO7060E: 装入共享库 %1 时遇到错误。返回码是 %2。

说明

装入共享库时遇到错误。

用户响应

确保共享库驻留在 PATH 或 LIBPATH 环境变量中指定的目录中。确保正确地构建了共享库。

EGL Java 运行时错误代码 CSO7080E

CSO7080E: 指定的协议 %1 无效。

说明

不识别链接中指定的协议。

用户响应

参考文档并指定有效的协议。

EGL Java 运行时错误代码 CSO7160E

CSO7160E: 在系统 %4 上的远程程序 %1 中，在日期 %2 时间 %3 时发生了错误。

说明

正在运行的 Java 程序调用指定的系统上的远程程序，而该远程程序在指定的日期和时间执行失败。

用户响应

有关问题分析的更详细描述，请检查远程服务器日志。

EGL Java 运行时错误代码 CSO7161E

CSO7161E: 由于在系统 %1 上尝试调用程序 %2 时发生应用程序错误，所以运行单元结束。%3

说明

执行远程程序时，在远程服务器上发生了导致远程运行单元异常终止的错误。在服务器作业日志中，位于此消息之前的诊断消息说明了错误的性质。如果有的话，可能随消息文本一起包括了其它信息。

用户响应

检查在服务器系统上记录的错误消息以确定修正原始问题所需执行的操作。

EGL Java 运行时错误代码 CSO7162E

CSO7162E: 在连接至系统 %1 时提供了无效的密码或用户标识。接收到的 Java 异常消息: %2。

说明

在连接至远程系统时提供的密码或用户标识未设置或无效。

用户响应

验证是否已设置连接。验证提供给远程系统的用户标识和密码是否正确，然后再试。

EGL Java 运行时错误代码 CSO7163E

CSO7163E: 对系统 %1 发生与用户 %2 相关的远程访问安全性错误。接收到的 Java 异常消息: %3

说明

当前连接至系统的指定用户不具有足够的权限或者不能访问指定系统上的远程资源。

用户响应

验证连接至远程机器的用户是否具有连接至远程机器和执行远程服务器程序所需的正确权限。

EGL Java 运行时错误代码 CSO7164E

CSO7164E: 对系统 %1 进行的远程连接出错。接收到的 Java 异常消息: %2

说明

与远程系统通信或连接至远程系统时发生错误。

用户响应

检查远程服务器是否可用; 然后重试。如果不能解决问题, 则与远程主机的系统管理员联系以确定实际问题。

EGL Java 运行时错误代码 CSO7165E

CSO7165E: 在系统 %1 上落实失败。%2

说明

落实操作在远程系统上失败。

用户响应

通过查看详细消息来诊断问题, 该消息在此处显示为 %2。

EGL Java 运行时错误代码 CSO7166E

CSO7166E: 在系统 %1 上回滚失败。%2

说明

回滚操作在远程系统上失败。

用户响应

通过查看详细消息来诊断问题, 该消息在此处显示为 %2。

EGL Java 运行时错误代码 CSO7360E

CSO7360E: 当在系统 %4 上调用程序 %3 时, 发生 AS400Toolbox 执行错误: %1, %2

说明

正在运行的 Java 程序或 applet 使用 Java400 协议来调用远程服务器程序。在尝试调用服务器程序时捕获了意外异常。此消息文本由 AS400 Toolbox 异常名称以及随后的与该异常一起返回的消息组成。

用户响应

使用提供的 AS400 Toolbox 错误消息用来分析问题的原因。

EGL Java 运行时错误代码 CSO7361E

CSO7361E: EGL OS/400® 主机服务错误。在系统 %1 上找不到必需的文件。

说明

正在运行的 Java 程序或 applet 使用 Java400 协议来调用远程服务器程序。当找不到远程捕捉器或者它不在服务器上正确的库中时，出现异常。

用户响应

检查是否在远程系统上正确安装了 EGL OS/400 主机服务。如果有最新的 PTF 可用，则应用它。

EGL Java 运行时错误代码 CSO7488E

CSO7488E: 未知 TCP/IP 主机名: %1

说明

尝试连接至远程 TCP/IP 侦听器程序时抛出了 UnknownHostException。

用户响应

执行下列操作:

- 将属性 `cs0.serverLinkage.xxx.location` 添加至运行时链接属性文件，其中 `xxx` 是被调用程序的名称或者是应用程序名，如有相关链接属性文件的 EGL 参考类型帮助页面所述。属性的值是有效 TCP/IP 主机名。
- 另外，在生成时设置 TCP/IP 主机名并重新生成程序：
 - 在链接选项部件中，在被调用程序的 `callLink` 元素中，将属性 **location** 设置为 TCP/IP 主机名
 - 如果您只想在运行时最终确定链接选项，则将属性 **remoteBind** 设置为 `RUNTIME`，并在将构建描述符选项 **genProperties** 设置为 `YES` 的情况下生成

有关其它详细信息，请参阅有关 `callLink` 元素、有关链接属性文件以及有关设置环境的 EGL 帮助页面。

EGL Java 运行时错误代码 CSO7489E

CSO7489E: 用来调用程序的链接信息不一致，或者缺少该信息。

说明

程序无法确定应该如何调用程序。

用户响应

提供所有必需的链接信息。必需的信息取决于期望执行的调用类型。参阅有关链接选项部件（尤其是有关 `callLink` 元素）的帮助页面。

EGL Java 运行时错误代码 CSO7610E

CSO7610E: 在调用 CICS ECI 以落实工作单元时遇到错误。CICS 返回码是 %1。

说明

客户机发出了落实请求，但未成功。在调用 CICS 外部调用接口以落实逻辑工作单元时遇到错误。

用户响应

请参阅适当的 CICS 文档以了解指定的错误的校正操作。

EGL Java 运行时错误代码 CSO7620E

CSO7620E: 在调用 CICS ECI 以回滚工作单元时遇到错误。CICS 返回码是 %1。

说明

客户机发出了回滚请求，但未成功。在调用 CICS 外部调用接口以回滚逻辑工作单元时遇到错误。

用户响应

请参阅适当的 CICS 文档以了解指定的错误的校正操作。

EGL Java 运行时错误代码 CSO7630E

CSO7630E: 在结束对 CICS 服务器进行的远程过程调用时遇到错误。CICS 返回码是 %1。

说明

在结束对 CICS 服务器进行的 EGL 远程过程调用之前，尝试了落实所有打开的逻辑工作单元，但未成功。此请求是通过 CICS 外部调用接口进行的。

用户响应

请参阅适当的 CICS 文档以了解指定的错误的校正操作。

EGL Java 运行时错误代码 CSO7640E

CSO7640E: 对于 `ctgport` 条目，%1 不是有效的值。

说明

`ctgport` 的值必须是整数。

用户响应

使用正确的 ctgport 号。

EGL Java 运行时错误代码 CSO7650E

CSO7650E: 在使用 **CICS ECI** 来调用程序 %1 时遇到错误。返回码: %2。CICS 系统标识: %3。

说明

当尝试调用远程服务器程序时, CICS 外部调用接口 (ECI) 函数调用返回了错误。

系统标识是服务器程序运行时所在的 CICS 系统的名称。如果系统标识是空白的, 则系统是在程序的 CICS 程序定义中指定的, 或者是在 CICS 客户机初始化文件中指定的。返回码是 CICS 返回码。

用户响应

更正返回码所指示的问题。

有关返回码的完整说明, 或者, 如果上面的内容未阐述该返回码, 则参阅系统的 CICS ECI 文档以获取有关校正操作的信息。

返回码值与 CICS ECI 包含文件 faaecih.h 或 cics_eci.h 中的符号相关联。

EGL Java 运行时错误代码 CSO7651E

CSO7651E: 在使用 **CICS ECI** 来调用程序 %1 时遇到错误。返回码: -3 (ECI_ERR_NO_CICS)。CICS 系统标识: %2。

说明

当尝试调用远程服务器程序时, CICS 外部调用接口 (ECI) 函数调用返回了错误。

系统标识是服务器程序运行时所在的 CICS 系统的名称。如果系统标识是空白的, 则系统是在程序的 CICS 程序定义中指定的, 或者是在 CICS 客户机初始化文件中指定的。返回码是 CICS 返回码。

CICS 返回码具有下列含义:

- -3 - ECI_ERR_NO_CICS

客户机或服务器系统不可用

用户响应

更正返回码所指示的问题。

有关返回码的完整说明, 或者, 如果上面的内容未阐述该返回码, 则参阅系统的 CICS ECI 文档以获取有关校正操作的信息。

返回码值与 CICS ECI 包含文件 faaecih.h 或 cics_eci.h 中的符号相关联。

EGL Java 运行时错误代码 CSO7652E

CSO7652E: 在使用 **CICS ECI** 来调用程序 **%1** 时遇到错误。返回码: **-4 (ECI_ERR_CICS_DIED)**。CICS 系统标识: **%2**。

说明

当尝试调用远程服务器程序时, CICS 外部调用接口 (ECI) 函数调用返回了错误。

系统标识是服务器程序运行时所在的 CICS 系统的名称。如果系统标识是空白的, 则系统是在程序的 CICS 程序定义中指定的, 或者是在 CICS 客户机初始化文件中指定的。返回码是 CICS 返回码。

CICS 返回码具有下列含义:

- -4 - ECI_ERR_CICS_DIED

服务器系统不再可用

用户响应

更正返回码所指示的问题。

有关返回码的完整说明, 或者, 如果上面的内容未阐述该返回码, 则参阅系统的 CICS ECI 文档以获取有关校正操作的信息。

返回码值与 CICS ECI 包含文件 faacih.h 或 cics_eci.h 中的符号相关联。

EGL Java 运行时错误代码 CSO7653E

CSO7653E: 在使用 **CICS ECI** 来调用程序 **%1** 时遇到错误。返回码: **-6 (ECI_ERR_RESPONSE_TIMEOUT)**。CICS 系统标识: **%2**。

说明

当尝试调用远程服务器程序时, CICS 外部调用接口 (ECI) 函数调用返回了错误。

系统标识是服务器程序运行时所在的 CICS 系统的名称。如果系统标识是空白的, 则系统是在程序的 CICS 程序定义中指定的, 或者是在 CICS 客户机初始化文件中指定的。返回码是 CICS 返回码。

CICS 返回码具有下列含义:

- -6 - ECI_ERR_RESPONSE_TIMEOUT

响应超时。时间限制是在环境变量 CSOTIMEOUT 中指定的。

用户响应

更正返回码所指示的问题。

有关返回码的完整说明, 或者, 如果上面的内容未阐述该返回码, 则参阅系统的 CICS ECI 文档以获取有关校正操作的信息。

返回码值与 CICS ECI 包含文件 faacih.h 或 cics_eci.h 中的符号相关联。

EGL Java 运行时错误代码 CSO7654E

CSO7654E: 在使用 **CICS ECI** 来调用程序 **%1** 时遇到错误。返回码: **-7 (ECI_ERR_TRANSACTION_ABEND)**。CICS 系统标识: **%2**。异常终止代码: **%3**。

说明

当尝试调用远程服务器程序时, CICS 外部调用接口 (ECI) 函数调用返回了错误。

系统标识是服务器程序运行时所在的 CICS 系统的名称。如果系统标识是空白的, 则系统是在程序的 CICS 程序定义中指定的, 或者是在 CICS 客户机初始化文件中指定的。返回码是 CICS 返回码。

CICS 返回码具有下列含义:

- **-7 - ECI_ERR_TRANSACTION_ABEND**

在服务器上发生异常终止。公共 ABEND 代码如下所示:

- AEIO - 服务器程序未定义
- AEI1 - 服务器事务未定义

用户响应

更正返回码所指示的问题。

有关返回码的完整说明, 或者, 如果上面的内容未阐述该返回码, 则参阅系统的 CICS ECI 文档以获取有关校正操作的信息。

返回码值与 CICS ECI 包含文件 `faecih.h` 或 `cics_eci.h` 中的符号相关联。

EGL Java 运行时错误代码 CSO7655E

CSO7655E: 在使用 **CICS ECI** 来调用程序 **%1** 时遇到错误。返回码: **-22 (ECI_ERR_UNKNOWN_SERVER)**。CICS 系统标识: **%2**。

说明

当尝试调用远程服务器程序时, CICS 外部调用接口 (ECI) 函数调用返回了错误。

系统标识是服务器程序运行时所在的 CICS 系统的名称。如果系统标识是空白的, 则系统是在程序的 CICS 程序定义中指定的, 或者是在 CICS 客户机初始化文件中指定的。返回码是 CICS 返回码。

CICS 返回码具有下列含义:

- **-22 - ECI_ERR_UNKNOWN_SERVER**

服务器系统未定义

用户响应

更正返回码所指示的问题。

有关返回码的完整说明，或者，如果上面的内容未阐述该返回码，则参阅系统的 CICS ECI 文档以获取有关校正操作的信息。

返回码值与 CICS ECI 包含文件 faacih.h 或 cics_eci.h 中的符号相关联。

EGL Java 运行时错误代码 CSO7656E

CSO7656E: 在使用 **CICS ECI** 来调用程序 **%1** 时遇到错误。返回码: **-27 (ECI_ERR_SECURITY_ERROR)**。CICS 系统标识: **%2**。

说明

当尝试调用远程服务器程序时，CICS 外部调用接口 (ECI) 函数调用返回了错误。

系统标识是服务器程序运行时所在的 CICS 系统的名称。如果系统标识是空白的，则系统是在程序的 CICS 程序定义中指定的，或者是在 CICS 客户机初始化文件中指定的。返回码是 CICS 返回码。

CICS 返回码具有下列含义:

- -27 - ECI_ERR_SECURITY_ERROR

用户标识或密码无效

用户响应

更正返回码所指示的问题。

有关返回码的完整说明，或者，如果上面的内容未阐述该返回码，则参阅系统的 CICS ECI 文档以获取有关校正操作的信息。

返回码值与 CICS ECI 包含文件 faacih.h 或 cics_eci.h 中的符号相关联。

EGL Java 运行时错误代码 CSO7657E

CSO7657E: 在使用 **CICS ECI** 来调用程序 **%1** 时遇到错误。返回码: **-28 (ECI_ERR_MAX_SYSTEMS)**。CICS 系统标识: **%2**。

说明

当尝试调用远程服务器程序时，CICS 外部调用接口 (ECI) 函数调用返回了错误。

系统标识是服务器程序运行时所在的 CICS 系统的名称。如果系统标识是空白的，则系统是在程序的 CICS 程序定义中指定的，或者是在 CICS 客户机初始化文件中指定的。返回码是 CICS 返回码。

CICS 返回码具有下列含义:

- -28 - ECI_ERR_MAX_SYSTEMS

已达到最大服务器数

用户响应

更正返回码所指示的问题。

有关返回码的完整说明，或者，如果上面的内容未阐述该返回码，则参阅系统的 CICS ECI 文档以获取有关校正操作的信息。

返回码值与 CICS ECI 包含文件 faaecih.h 或 cics_eci.h 中的符号相关联。

EGL Java 运行时错误代码 CSO7658E

CSO7658E: 在系统 %2 上为用户 %3 调用程序 %1 时遇到错误。CICS ECI 调用返回了 RC %4 和异常终止代码 %5。

说明

代表消息所标识的用户从网关对指定系统进行的 CICS ECI 调用返回了非零返回码。

用户响应

更正返回码所指示的问题。

有关返回码的完整说明，参阅系统的 CICS ECI 文档以获取有关校正操作的信息。

返回码值与 CICS ECI 包含文件 faaecih.h 或 cics_eci.h 中的符号相关联。

EGL Java 运行时错误代码 CSO7659E

CSO7659E: 对 CICS 系统 %1 的 ECI 请求流发生异常。异常: %2

说明

当尝试将 ECI 请求从网关发送到消息中标识的 CICS 系统时，在流方法中发生意外异常。

用户响应

检查返回的异常字符串。如果无法根据异常确定问题的原因，请与 IBM 支持机构联系以获取帮助。

EGL Java 运行时错误代码 CSO7669E

CSO7669E: 连接至 CTG 时遇到错误。CTG 位置: %1, CTG 端口: %2。异常: %3

说明

连接至 CICS 事务网关时发生意外异常。

用户响应

检查返回的异常字符串。如果无法根据异常确定问题的原因，请与 IBM 支持机构联系以获取帮助。

EGL Java 运行时错误代码 CSO7670E

CSO7670E: 与 CTG 断开连接时遇到错误。CTG 位置: %1, CTG 端口: %2。异常: %3

说明

与 CICS 事务网关断开连接时发生意外异常。

用户响应

检查返回的异常字符串。如果无法根据异常确定问题的原因，请与 IBM 支持机构联系以获取帮助。

EGL Java 运行时错误代码 CSO7671E

CSO7671E: 当使用 **CICSSSL** 协议时，必须同时指定 **ctgKeyStore** 和 **ctgKeyStorePassword**。

说明

未指定必需值，因此无法完成调用。

用户响应

确保同时指定了 **ctgKeyStore** 和 **ctgKeyStorePassword**。

EGL Java 运行时错误代码 CSO7816E

CSO7816E: 对于用户标识 %4，当网关尝试连接至具有主机名 %1 端口 %2 的服务器时，发生套接字异常。异常是: %3。

说明

用于创建套接字并将该套接字从网关连接至消息中标识的服务器系统的套接字调用失败，并抛出了显示的异常。

对于一个服务器调用，EGL 网关尝试了用于创建和连接 TCP/IP 套接字的套接字调用。该套接字调用失败，并抛出消息中指示的异常。

用户响应

检查异常信息以确定从网关进行的套接字调用的失败原因。如果无法通过检查异常信息来确定问题的原因，请与 IBM 支持机构联系以获取帮助。

EGL Java 运行时错误代码 CSO7819E

CSO7819E: 函数 %2 发生意外异常。异常: %1

说明

EGL 网关从消息中标识的函数接收到意外异常。可能发生了内部错误。

用户响应

如果无法通过检查异常信息来确定问题的根源，请与 IBM 支持机构联系以获取帮助。

EGL Java 运行时错误代码 CSO7831E

CSO7831E: 客户机的缓冲区对于进行调用时传递的数据量而言太小。确保正在传递的参数的累计大小不超过允许的最大大小（即 **32567** 个字节）。

说明

无法使客户机建立的缓冲区与正被传递给远程被调用程序的参数的累计大小一样大。

用户响应

确保正在传递的参数的累计大小不超过允许的最大大小（即 32567 个字节）。如果它们未超过最大大小，并且发生了此错误，请向 IBM 支持中心报告此错误。

EGL Java 运行时错误代码 CSO7836E

CSO7836E: 客户机已接收到一个通知，指示服务器未能启动远程被调用程序。原因码：%1。

说明

服务器未能运行远程被调用程序，并且返回了原因码以便于确定问题。

用户响应

原因码如下所示：

- 2 - 服务器未能装入被调用程序的类。服务器跟踪文件可能显示了更具体的信息。确保类可供服务器使用。

此问题可能是由于不正确转换传递给服务器的类名引起的。请查看有关数据转换的帮助页面，以验证链接选项部件、被调用程序的 callLink 元素、属性 conversionTable 中是否指定了正确的转换表。

- 3 - 被调用程序由于错误而结束。服务器跟踪文件可能显示了更具体的信息。

对于以上未列示的任何原因码，或者如果您无法确定发生故障的原因，请与 IBM 支持机构联系。

EGL Java 运行时错误代码 CSO7840E

CSO7840E: 客户机从服务器接收到一个通知，指示远程被调用程序失败，返回码为 %1。

说明

远程被调用程序已运行，但是以非零返回码结束。是程序而不是通信出了问题。

用户响应

检查或跟踪被调用程序，以确定它以非零返回码结束的原因。

EGL Java 运行时错误代码 CSO7885E

CSO7885E: 当代表用户标识 %2 对主机名 %1 进行调用时, TCP/IP 读函数失败。
返回的异常是: %3

说明

当尝试执行 TCP/IP 读函数时, EGL 网关接收到异常。

用户响应

检查返回的异常信息以确定问题的原因。如果无法确定故障的发生原因, 请与 IBM 支持机构联系以获取帮助。

EGL Java 运行时错误代码 CSO7886E

CSO7886E: 当代表用户标识 %2 对主机名 %1 进行调用时, TCP/IP 写函数失败。
返回的异常是: %3

说明

当尝试执行 TCP/IP 写函数时, EGL 网关接收到异常。

用户响应

检查返回的异常信息以确定问题的原因。如果无法确定故障的发生原因, 请与 IBM 支持机构联系以获取帮助。

EGL Java 运行时错误代码 CSO7955E

CSO7955E: %1, %2

说明

捕获了意外的 Java 异常。

该消息文本显示 Java 异常的名称, 后跟与该异常一起抛出的 Java 消息。

用户响应

查看消息并适当地作出响应。

EGL Java 运行时错误代码 CSO7957E

CSO7957E: 转换表名 %1 对 Java 数据转换无效。

说明

您正在使用生成的 Java 类来调用程序, 并且未正确地指定转换表来将 Java 数据转换为被调用程序使用的格式。

用户响应

查看有关数据转换的帮助页面以确定转换表名，该名称是在链接选项部件、被调用程序的 `callLink` 元素和属性 `conversionTable` 中指定的。

EGL Java 运行时错误代码 CSO7958E

CSO7958E: 本机代码未向 Java 包装器提供类型为 `CSOPowerServer` 的对象，而在 Java 包装器与 EGL 生成程序之间转换数据需要该对象。

说明

在没有先实例化类 `CSOPowerServer` 的对象并向 Java 包装器提供该对象的情况下，本机 Java 代码调用了 Java 包装器的调用或执行方法。

用户响应

查看有关 Java 包装器的帮助页面以获取有关访问 EGL 中间件的详细信息，该中间件始终是进行数据转换所必需的。

EGL Java 运行时错误代码 CSO7966E

CSO7966E: 找不到转换表 %2 的代码页编码 %1。

说明

链接选项中指定的转换表需要一种编码，而该编码在正在使用的 Java 虚拟机 (JVM) 中不可用。

用户响应

查看有关数据转换的帮助页面以确定正确的转换表名，该名称是在链接选项部件、被调用程序的 `callLink` 元素和属性 `conversionTable` 中指定的。如果指定了正确的转换表，则确保正在使用的 JVM 受 EGL 的 Java 运行时环境支持。

如果前面的步骤没有发现问题，则考虑是 JVM 的安装有缺陷还是 Java 虚拟机不支持所有编码。在这些情况下，请参阅 JVM 供应商的文档，或者与 JVM 供应商联系以获取帮助。

如果在浏览器中运行 applet 客户机时遇到了错误，则是在客户机 applet 使用的 `PowerServer SessionManager` 中发生了错误。在这种情况下，请参阅正在运行 `SessionManager` 的 JVM 的文档，或者与 JVM 供应商联系。

EGL Java 运行时错误代码 CSO7968E

CSO7968E: `Host %1` 是未知的，或者找不到它。

说明

未在链接中指定远程系统。

用户响应

必须在链接部件的位置字段中指定远程系统。

EGL Java 运行时错误代码 CSO7970E

CSO7970E: 未能装入必需的 EGL 共享库 %1, 原因: %2

说明

该共享库是完成操作所必需的, 但未能装入该共享库。

用户响应

确保共享库位于系统上。它必须包括在指定共享库路径的环境变量 PATH 或 LIBPATH 中。

EGL Java 运行时错误代码 CSO7975E

CSO7975E: 无法打开属性文件 %1。

说明

无法打开程序所需的属性文件。当启动程序时, 可在命令行上指定属性文件的名称。如果启动程序时未给定名称, 则缺省情况下使用以下名称:

```
tcpilistener.properties
```

属性文件不存在, 或者虽然属性文件存在但是无法将其打开。

用户响应

确保属性文件存在, 并且程序具有读取它所需的正确许可权, 然后再次运行该程序。

EGL Java 运行时错误代码 CSO7976E

CSO7976E: 无法打开跟踪文件 %1。异常为 %2, 消息如下所示: %3。

说明

当程序尝试打开跟踪输出文件时发生了异常。

用户响应

更正问题并重新运行程序。

EGL Java 运行时错误代码 CSO7977E

CSO7977E: 程序属性文件未包含 %1 属性的有效设置, 该属性是必需的。

说明

未在程序属性文件中定义该属性。

用户响应

将该属性添加至程序属性文件，并重新运行程序。有关详细信息，请参阅有关 Java 运行时属性的帮助页面。

EGL Java 运行时错误代码 CSO7978E

CSO7978E: 发生了意外异常。异常为 %1，消息如下所示: %2。

说明

程序遇到了错误。

用户响应

更正问题并重新运行程序。

EGL Java 运行时错误代码 CSO7979E

CSO7979E: 无法创建 `InitialContext`。异常为 %1

说明

从 `javax.naming.InitialContext` 的构造函数中抛出了异常。程序需要创建 `InitialContext` 对象以访问 J2EE 环境设置。

用户响应

使用异常的文本和 J2EE 环境的文档来更正问题。

EGL Java 运行时错误代码 CSO8000E

CSO8000E: 对网关输入的密码已到期。%1

说明

当尝试使用提供的密码授权用户时，EGL GatewayServlet 接收到密码已到期异常。

用户响应

检查返回的异常信息以确定问题的原因。通过提供新密码来更正问题。

EGL Java 运行时错误代码 CSO8001E

CSO8001E: 对网关输入的密码无效。%1

说明

当尝试通过提供的密码来认证用户时，EGL GatewayServlet 接收到密码无效异常。

用户响应

检查返回的异常信息以确定问题的原因。通过提供新密码来更正问题。

EGL Java 运行时错误代码 CSO8002E

CSO8002E: 对网关输入的用户标识无效。%1

说明

当尝试通过提供的用户标识来认证用户时，EGL GatewayServlet 接收到用户标识无效异常。

用户响应

检查返回的异常信息以确定问题的原因。通过提供新用户标识来更正问题。

EGL Java 运行时错误代码 CSO8003E

CSO8003E: 对应 %1 的为空条目

说明

检测到空条目。

用户响应

检查返回的异常信息以确定问题的原因。通过提供必需的条目来更正问题。

EGL Java 运行时错误代码 CSO8004E

CSO8004E: 网关接收到未知的安全性错误。

说明

当尝试通过提供的用户信息来认证用户时，EGL GatewayServlet 接收到安全性未知异常。

用户响应

检查返回的异常信息以确定问题的原因。通过提供新用户信息来更正问题。如果无法确定故障的发生原因，请与 IBM 支持机构联系以获取帮助。

EGL Java 运行时错误代码 CSO8005E

CSO8005E: 在更改密码时发生错误。%1

说明

当尝试更改提供的密码时，EGL GatewayServlet 接收到错误。

用户响应

检查返回的异常信息以确定问题的原因。通过提供新密码来更正问题。如果无法确定故障的发生原因，请与 IBM 支持机构联系以获取帮助。

EGL Java 运行时错误代码 CSO8100E

CSO8100E: 未能获取连接工厂。异常为 %1

说明

当 remoteComType 属性值为 CICSJ2C 时，在查找对调用使用的连接工厂期间抛出了异常。remoteComType 属性在链接选项部件和被调用程序的 callLink 元素中。

连接工厂的名称以 java:comp/env/ 开头，后面跟着您在同一 callLink 元素的 location 属性中设置的值。

用户响应

确保在 J2EE 环境中正确定义了连接工厂，并且被调用程序的 callLink 元素中的 location 属性值是正确的。

EGL Java 运行时错误代码 CSO8101E

CSO8101E: 无法获取连接。异常为: %1

说明

当 remoteComType 属性值为 CICSJ2C 时，用来执行调用的 ConnectionFactory 对象的 getConnection 方法抛出了异常。remoteComType 属性在链接选项部件和被调用程序的 callLink 元素中。

用户响应

可能未正确地定义或配置连接工厂或资源适配器。通过参考异常文本、资源适配器文档和 J2EE 环境文档来诊断问题。

EGL Java 运行时错误代码 CSO8102E

CSO8102E: 无法获取交互。异常为: %1

说明

当 remoteComType 属性值为 CICSJ2C 时，用来执行调用的 Connection 对象的 createInteraction 方法抛出了异常。remoteComType 属性在链接选项部件和被调用程序的 callLink 元素中。

用户响应

可能未正确地定义或配置连接工厂或资源适配器。通过使用异常文本、资源适配器文档和 J2EE 环境文档来诊断问题。

EGL Java 运行时错误代码 CSO8103E

CSO8103E: 无法设置交互查询描述。异常为 %1

说明

当 remoteComType 属性值为 CICSJ2C 时，用来执行调用的 ECIInteractionSpec 对象的 setInteractionVerb 方法抛出了异常。remoteComType 属性在链接选项部件和被调用程序的 callLink 元素中。

用户响应

可能未正确地定义或配置连接工厂或资源适配器。通过使用异常文本、资源适配器文档和 J2EE 环境文档来诊断问题。

EGL Java 运行时错误代码 CSO8104E

CSO8104E: 尝试与 CICS 通信时发生了错误。异常为 %1

说明

当 remoteComType 属性值为 CICSJ2C 时，用来执行调用的交互对象的 execute 方法抛出了异常。remoteComType 属性在链接选项部件和被调用程序的 callLink 元素中。

用户响应

可能未正确地定义或配置连接工厂或资源适配器。通过使用异常文本、资源适配器文档和 J2EE 环境文档来诊断问题。网关日志或者远程系统上的日志文件中可能提供了其它信息。

EGL Java 运行时错误代码 CSO8105E

CSO8105E: 无法关闭交互或连接。异常为 %1

说明

当 remoteComType 属性值为 CICSJ2C 时，用来执行调用的连接或交互对象的 close 方法抛出了异常。remoteComType 属性在链接选项部件和被调用程序的 callLink 元素中。

用户响应

可能未正确地定义或配置连接工厂或资源适配器。通过使用异常文本、资源适配器文档和 J2EE 环境文档来诊断问题。

EGL Java 运行时错误代码 CSO8106E

CSO8106E: 无法获取客户机工作单元的 LocalTransaction。异常为 %1

说明

在以下情况下用来执行调用的 Connection 对象的 getLocalTransaction 方法抛出了异常。

- remoteComType 属性值为 CICSJ2C
- luwControl 属性值为 CLIENT

这些属性都在链接选项部件和被调用程序的 callLink 元素中。

用户响应

可能未正确地定义或配置连接工厂或资源适配器。通过使用异常文本、资源适配器文档和 J2EE 环境文档来诊断问题。

EGL Java 运行时错误代码 CSO8107E

CSO8107E: 无法对 CICSJ2C 调用设置超时值。异常为 %1

说明

当 remoteComType 属性值为 CICSJ2C 时，用来执行调用的 ECIInteractionSpec 对象的 setExecuteTimeout 方法抛出了异常。remoteComType 属性在链接选项部件和被调用程序的 callLink 元素中。

用户响应

可能未正确地定义或配置连接工厂或资源适配器。通过使用异常文本、资源适配器文档和 J2EE 环境文档来诊断问题。

EGL Java 运行时错误代码 CSO8108E

CSO8108E: 尝试与 CICS 通信时发生了错误。

说明

用来执行调用的交互对象的 execute 方法返回了 false。未成功完成调用。

用户响应

可能未正确地定义或配置连接工厂或资源适配器。通过使用异常文本、资源适配器文档和 J2EE 环境文档来诊断问题。网关日志或者远程系统上的日志文件中可能提供了其它信息。

EGL Java 运行时错误代码 CSO8109E

CSO8109E: 超时值 %1 无效。它必须是数字。

说明

对超时指定了无效的值。

用户响应

不要指定超时值，或者指定一个数字。

EGL Java 运行时错误代码 CSO8110E

CSO8110E: 由于至少一个参数是动态数组，所以必须将 parmForm 链接属性设置为 COMMPTR 才能调用程序 %1。

说明

由于其中一个参数是动态数组，所以 parmForm 必须是 COMMPTR。

用户响应

将 parmForm 更改为 COMMPTR。

EGL Java 运行时错误代码 CSO8180E

CSO8180E: 链接指定了 J2EE 服务器中的 DEBUG 调用。未在 J2EE 服务器上运行该调用，J2EE 服务器未处于调试方式，或者 J2EE 服务器未启用 EGL 调试。

说明

无法完成 DEBUG 调用。

用户响应

如果未在 J2EE 服务器上运行该调用，则必须在链接的 location 字段中指定运行 EGL 调试器的机器的 TCP/IP 主机名。如果正在 J2EE 服务器上运行该调用，则确保它是以前以调试方式启动的，并确保已将 EGL 调试器 jar 文件添加至其中。

EGL Java 运行时错误代码 CSO8181E

CSO8181E: 无法与位于主机名 %1 端口 %2 处的 EGL 调试器取得联系。异常是 %3

说明

由于无法联系 EGL 调试器，所以无法完成 DEBUG 调用。

用户响应

确保正在位于指定主机名和端口处的 EGL 调试器中运行 EGL 侦听器。

EGL Java 运行时错误代码 CSO8182E

CSO8182E: 当与位于主机名 %1 端口 %2 处的 EGL 调试器通信时发生错误。异常是 %3

说明

EGL 调试器与调用程序之间的通信失败。

用户响应

使用异常消息中的信息来更正问题。

EGL Java 运行时错误代码 CSO8200E

CSO8200E: 无法将数组包装器 %1 扩展为超出它的最大大小。此错误是在方法 %2 中发生的。

说明

超出数组的最大大小。

用户响应

在尝试对数组进行添加之前，检查数组的大小和最大大小。

EGL Java 运行时错误代码 CSO8201E

CSO8201E: 对于数组包装器 %2, %1 不是有效的下标。最大大小: %3。当前大小: %4

说明

下标超出数组的边界。

用户响应

使用有效的下标。

EGL Java 运行时错误代码 CSO8202E

CSO8202E: 对于数组包装器 %2, %1 不是有效的最大大小。

说明

属性 `maxSize` 必须大于等于零。

用户响应

不要将属性 `maxSize` 设置为负数。

EGL Java 运行时错误代码 CSO8203E

CSO8203E: %1 是无效的对象类型，不能将其添加至类型为 %2 的数组包装器。

说明

数组的内容必须与其定义匹配。

用户响应

更改数组所存储的对象的类型，或者不要尝试在数组中存储该类型的对象。

EGL Java 运行时错误代码 CSO8204E

CSO8204E: 不能将 `Any`、`Dictionary`、`ArrayDictionary`、`Blob`、`Clob` 或 `Ref` 变量作为参数传递。

说明

列示的类型可能不能用作调用语句中的参数。此外，包含列示类型的类型可能不能用作参数。

用户响应

不要将该种类型的参数传递至被调用程序。

EGL Java 运行时错误代码 EGL0650E

EGL0650E: %1RequestAttr 函数对键 %2 失败。错误: %3

说明

当使用给定的键进行调用时，EGL GetRequestAttr 或 SetRequestAttr 函数失败。

用户响应

使用此消息的错误部分来诊断和更正问题。确保该函数是在 PageHandler 函数中使用的。

EGL Java 运行时错误代码 EGL0651E

EGL0651E: %1SessionAttr 函数对键 %2 失败。错误: %3

说明

当使用给定的键进行调用时，EGL GetSessionAttr 或 SetSessionAttr 函数失败。

用户响应

使用此消息的错误部分来诊断和更正问题。确保该函数是在 PageHandler 函数中调用的。

EGL Java 运行时错误代码 EGL0652E

EGL0652E: forward 语句对标号 %1 失败。错误: %2

说明

未能将控制权转发至给定的标号。

用户响应

使用此消息的错误部分来诊断和更正问题。确保正确生成了与该标号相关联的 EGL 对象，并确保在应用程序配置文件中定义了该标号。

EGL Java 运行时错误代码 EGL0653E

EGL0653E: 未能根据 EGL 对象 %1 来创建 Bean。错误: %2

说明

未能根据 EGL 记录或 PageHandler 定义来创建访问 bean。

用户响应

使用此消息的错误部分来诊断和更正问题。

EGL Java 运行时错误代码 EGL0654E

EGL0654E: SetError 函数对项 %1 键 %2 失败。错误: %3

说明

当使用给定的消息键进行调用时，SetError 函数失败。

用户响应

使用此消息的错误部分来诊断和更正问题。确保项在 JSP 中有错误条目，并确保已在消息资源文件中定义该键。

EGL Java 运行时错误代码 EGL0655E

EGL0655E: 未能将数据从 Bean 复制到 EGL 记录 %1。错误: %2

说明

尝试将数据从表单 bean 移至记录时失败。

用户响应

使用此消息的错误部分来诊断和更正问题。确保 bean 定义与记录定义匹配。

EGL Java 运行时错误代码 EGL0656E

EGL0656E: 无法将大小为 %1 的数组赋值给大小为 %2 的静态数组。

说明

数组的大小必须匹配。

用户响应

检查 EGL 数组定义并确保数组大小相同。

EGL Java 运行时错误代码 EGL0657E

EGL0657E: onPageLoad 参数的处理失败。错误: %1。

说明

当 EGL 尝试将值接收到 onPageLoad 函数的参数中时发生错误。

用户响应

使用此消息的错误部分来诊断和更正问题。确保传递的值的类型定义与对 onPageLoad 函数中的参数定义的类型匹配。

EGL Java 运行时错误代码 VGJ0001E

VGJ0001E: %1 最大值溢出。

说明

在算术计算期间，一个值被零除，或者是中间结果超出 18 个有效位。除非系统变量 **VGVar.handleOverflow** 设置为 2，否则程序会结束。

用户响应

执行下列一项或多项操作：

- 更正程序的逻辑以避免发生错误。
- 定义程序逻辑以处理溢出条件；使用系统变量 **VGVar.handleOverflow** 和 **overflowIndicator**。

EGL Java 运行时错误代码 VGJ0002E

VGJ0002E: 发生了错误 %1。在消息文件 %2 中找不到此错误的消息文本。

说明

消息文件可能已损坏，或者是来自 EGL 的较早发行版。

用户响应

完成下列其中一项指示信息：

- 如果已经从文件 `fda.jar` 解压缩类文件，则验证您拥有的类是否与该 JAR 文件中的类处于同一发行版或维护级别。如果您发现不匹配，则将旧的类替换为正确版本。
- 从 EGL 重新安装 `fda6.jar`。

如果问题仍然存在，则执行以下操作：

1. 记录消息号和消息文本。

注： 错误消息包括下列重要信息：

- 错误的发生位置
 - 内部错误的类型
2. 记录发生此消息的情况。
 3. 有关如何向“IBM 支持中心”报告可能存在的缺陷的更多指示信息，参阅产品安装手册。

EGL Java 运行时错误代码 VGJ0003E

VGJ0003E: 在位置 %1 处发生了内部错误。

说明

仅当未满足系统约束或需求，或者不正确地使用了 EGL 程序部件时，才会发生此错误。在错误中指定的位置仅供 IBM 用于诊断。

用户响应

检查程序设置，并重新启动系统。如果问题仍然存在，则执行以下操作：

1. 记录消息号和消息文本。

注： 错误消息包括下列重要信息：

- 错误的发生位置
- 内部错误的类型

2. 记录发生此消息的情况。
3. 有关如何向“IBM 支持中心”报告可能存在的缺陷的更多指示信息，参阅产品安装手册。

EGL Java 运行时错误代码 VGJ0004I

VGJ0004I: 在 %1 函数 %2 中发生了错误。

说明

当发生错误时，此消息伴随着另一条消息。此消息标识发生了错误的程序或记录，以及当时正在执行的函数。

用户响应

无。

EGL Java 运行时错误代码 VGJ0005I

VGJ0005I: 在 %1 中发生了错误。

说明

此消息伴随着另一条消息，它标识发生了错误的程序或记录。

用户响应

无。

EGL Java 运行时错误代码 VGJ0006E

VGJ0006E: 在执行 I/O 操作期间发生了错误。%1

说明

I/O 操作失败，EGL 语句没有用于处理错误的 try 语句。

用户响应

如果要想程序处理错误，请将 `handleHardIOErrors` 设置为 1 并将 I/O 语句放置在 try 语句中，如下示例所示：

```
VGVar.handleHardIOErrors = 1;

if (userRequest == "A")
```

```
try
  add record1
onException
  myErrorHandler(12);
end
end
```

EGL Java 运行时错误代码 VGJ0007E

VGJ0007E: %1 中的最小值溢出。

说明

算术运算产生的结果超出数据类型允许的最小值。

用户响应

对算术表达式作相应调整。

EGL Java 运行时错误代码 VGJ0008E

VGJ0008E: 发生了可恢复资源错误。%1

说明

关闭、落实或回滚可恢复资源时发生了错误。

用户响应

使用错误消息中的信息来更正该问题。

EGL Java 运行时错误代码 VGJ0009E

VGJ0009E: 在 %2 中找不到带有标识 %1 的字段。

说明

因为指定字段不存在，所以动态访问失败。

用户响应

不要访问不存在的字段。

EGL Java 运行时错误代码 VGJ0010E

VGJ0010E: 对 %1 赋值失败：赋值源 %2 不兼容。

说明

源的类型不是可指定给目标的类型。

用户响应

确保在赋值时源和目标类型兼容。

EGL Java 运行时错误代码 VGJ0011E

VGJ0011E: 不能将值 %1 解析为基本类型。

说明

变量被用作数据项，但它不是数据项。

用户响应

更改程序以使它不会将该变量用作数据项。

EGL Java 运行时错误代码 VGJ0012E

VGJ0012E: 未能对算术表达式求值: %1 中的类型不兼容。

说明

表达式中的值的类型不兼容。

用户响应

更改程序以在表达式中使用兼容类型。

EGL Java 运行时错误代码 VGJ0013E

VGJ0013E: set 语句失败: %1 不能设置为 %2 状态。

说明

该变量不支持指定状态。

用户响应

更改程序以使它不会尝试此操作。

EGL Java 运行时错误代码 VGJ0014E

VGJ0014E: 不能对 %1 指定下标。它不是数组。

说明

变量被用作数组，但它不是数组。

用户响应

更改程序以使它不会将变量用作数组。

EGL Java 运行时错误代码 VGJ0015E

VGJ0015E: %1, %2

说明

发生了错误。异常及其消息将插入到此消息中。

用户响应

使用消息插入内容中的信息来更正该问题。

EGL Java 运行时错误代码 VGJ0016E

VGJ0016E: 未对任何变量 %1 指定值。

说明

在对该变量赋值之前使用了该变量。

用户响应

更改程序以在使用变量之前对它赋值。

EGL Java 运行时错误代码 VGJ0017E

VGJ0017E: 引用变量 %1 为 Nil。

说明

该变量必须引用值才能使用。

用户响应

在使用该变量之前对其赋值。

EGL Java 运行时错误代码 VGJ0018E

VGJ0018E: 不能对结构化记录 %1 执行动态访问。

说明

结构化记录不允许动态访问。

用户响应

不要对结构化记录使用动态访问。

EGL Java 运行时错误代码 VGJ0019E

VGJ0019E: 不能复制 %1。

说明

尝试复制可能不应复制的内容；或者尝试复制失败。

用户响应

EGL Java 运行时错误代码 VGJ0020E

VGJ0020E: 名为 %1 的变量不能用作 %2。

说明

该变量的类型不允许该变量用作指定类型。

用户响应

更改程序以使它不会将该变量用作不同类型。

EGL Java 运行时错误代码 VGJ0021E

VGJ0021E: 不能针对 %2 状态测试 %1。

说明

IS 或 NOT 表达式中发生了此错误。表达式左边的变量不支持在表达式右边指定的状态。

用户响应

除去或修改表达式。

EGL Java 运行时错误代码 VGJ0050E

VGJ0050E: 装入程序 %1 时发生了异常。异常: %2 消息: %3

说明

未能装入程序的类。

用户响应

使用异常消息来诊断和修正问题。发生此错误最常见的原因是包含程序类文件的 jar 文件或目录未列示在 CLASSPATH 环境变量中。

EGL Java 运行时错误代码 VGJ0055E

VGJ0055E: 调用程序 %1 时发生了错误。错误代码为 %2 (%3)。

说明

在调用本地 Java 程序期间发生了错误。

用户响应

使用异常消息来诊断和修正问题。

EGL Java 运行时错误代码 VGJ0056E

VGJ0056E: 被调用程序 %1 期望是 %2 参数, 但是传递的却是 %3。

说明

传递给被调用程序的参数数目是错误的。

用户响应

重新编写调用程序或被调用程序, 以便使它们期望传递相同数目的参数。

EGL Java 运行时错误代码 VGJ0057E

VGJ0057E: 在将参数传递给被调用程序 %1 时发生了异常。异常: %2 消息: %3

说明

调用 Java 程序时发生了错误。在程序开始之前或之后可能已经发生了错误。

用户响应

使用异常及其消息来诊断和修正问题。

EGL Java 运行时错误代码 VGJ0058E

VGJ0058E: 未能装入属性文件 %1。

说明

未能装入程序的属性文件。属性文件的名称是从系统属性 `vgj.properties.file` 获取的。

用户响应

确保 `vgj.properties.file` 具有正确的文件名, 并确保该属性文件位于列示在 `CLASSPATH` 环境变量中的 Jar 文件或目录中。

EGL Java 运行时错误代码 VGJ0060E

VGJ0060E: 对类 %1 执行的 `StartTransaction` 失败。异常为 %2。

说明

当程序尝试启动新的 JVM 以将指定的服务器类作为新事务运行时, 抛出了异常。属性 `vgj.java.command` 指定了用来启动新的 JVM 的命令。缺省命令是 `java`。

用户响应

确保属性 `vgj.java.command` 具有正确的值, 并确保程序具有创建新进程所需的许可权。

将 `startTransaction` 语句放置在 `try` 语句内以防止此错误成为致命错误。当 `startTransaction` 在 `try` 语句内失败时, 将把一个错误代码存储在 `errorCode` 系统变量中。

EGL Java 运行时错误代码 VGJ0062E

VGJ0062E: 传递给 MQ 程序 %1 的一个或多个参数具有错误类型。%2

说明

尝试调用 MQ 程序时抛出了异常。参数不正确。

用户响应

参阅 MQ 程序的文档和异常的消息来更正错误。

EGL Java 运行时错误代码 VGJ0064E

VGJ0064E: 程序 %1 期望文本表单 %2, 但它在 show 语句上得到文本表单 %3。

说明

两个程序必须使用同一个文本表单。

用户响应

将程序修改为使用同一个文本表单并重新生成。

EGL Java 运行时错误代码 VGJ0100E

VGJ0100E: 数据 %1 的格式不是 %2。

说明

该项中的数据采用了意外的格式。另一个项可能改写了指定的项。

用户响应

更正程序逻辑以避免发生错误。

EGL Java 运行时错误代码 VGJ0104E

VGJ0104E: %1 不是 %3 的下标 %2 的有效下标。

说明

与多维数组配合使用的其中一个下标无效。下标值必须在 1 与对带有下标的项定义的出现次数之间。

用户响应

确保下标值是带有下标的项的有效下标。

EGL Java 运行时错误代码 VGJ0105E

VGJ0105E: %1 不是 %2 的有效下标。

说明

下标值必须在 1 与为带有下标的项定义的出现次数之间。

用户响应

确保下标值是带有下标的项的有效下标。

EGL Java 运行时错误代码 VGJ0106E

VGJ0106E: 在将 %1 赋值给 %2 时发生了用户溢出。

说明

不截断有效位，则赋值的目标不足以容纳结果。系统变量 `VGVar.handleOverflow` 的值为 1，它将导致程序结束。

用户响应

执行下列操作：

- 增大目标中的有效位数；或者
- 定义程序逻辑以处理溢出条件；使用系统变量 `VGVar.handleOverflow` 和 `overflowIndicator`。

EGL Java 运行时错误代码 VGJ0108E

VGJ0108E: 为 HEX 项 %1 指定了非十六进制值 %2。

说明

HEX 项只能接收十六进制数字。

用户响应

确保源值只包括十六进制数字。

EGL Java 运行时错误代码 VGJ0109E

VGJ0109E: 从 %2 中为 HEX 项 %1 指定了非十六进制值: %3。

说明

HEX 项只能接收十六进制数字。

用户响应

确保赋值中的源只包含十六进制数字。

EGL Java 运行时错误代码 VGJ0110E

VGJ0110E: HEX 项 %1 与非十六进制值 %2 进行了比较。

说明

HEX 项只能与十六进制数字进行比较。

用户响应

确保比较值只包括十六进制数字。

EGL Java 运行时错误代码 VGJ0111E

VGJ0111E: 从 %2 中将 HEX 项 %1 与非十六进制值进行了比较: %3。

说明

HEX 项只能与十六进制数字进行比较。

用户响应

确保比较值只包含十六进制数字。

EGL Java 运行时错误代码 VGJ0112E

VGJ0112E: 为 NUM 项 %1 指定了非数值: %2。

说明

只能为 NUM 项指定数值。这样的值包含数字，并且可以具有前导和结尾空格、小数点和前导符号。允许两个位之间具有小数点，可以刚好在第一位前面，或者紧接着最后一位的后面。

用户响应

确保源值是数字。

EGL Java 运行时错误代码 VGJ0113E

VGJ0113E: 从 %2 中为 NUM 项 %1 指定了非数值: %3。

说明

只能为 NUM 项指定数值。这样的值包含数字，并且可以具有前导和结尾空格、小数点和前导符号。允许两个位之间具有小数点，可以刚好在第一位前面，或者紧接着最后一位的后面。

用户响应

确保源值是数字。

EGL Java 运行时错误代码 VGJ0114E

VGJ0114E: 项 %1 (%2) 的值作为下标是无效的。

说明

对于数组中的任何元素，该值具有太多位数，不能作为下标。下标值必须介于 1 与对结构项声明的 `occurs` 数目之间。

用户响应

确保下标值为数组的有效下标。

EGL Java 运行时错误代码 VGJ0115E

VGJ0115E: 不能将字符串赋予 %1。字符串为 %2。

说明

不能将字符串赋予该项。

用户响应

不要对该项指定字符串。

EGL Java 运行时错误代码 VGJ0116E

VGJ0116E: 不能将数字赋予 %1。数字为 %2。

说明

不能将数字赋予该项。

用户响应

不要对该项指定数字。

EGL Java 运行时错误代码 VGJ0117E

VGJ0117E: 不能将 %1 转换为 `long` 类型。

说明

不能将该项转换为 `long` 类型。

用户响应

完成下列步骤:

1. 记录消息号和消息文本。

注: 错误消息包括下列重要信息:

- 错误的发生位置
- 内部错误的类型

2. 记录发生此消息的情况。

3. 有关如何向“IBM 支持中心”报告可能存在的缺陷的更多指示信息，参阅 *EGL Installation Guide*。

EGL Java 运行时错误代码 VGJ0118E

VGJ0118E: %1 不能转换为数字。

说明

该项不能转换为数字。

用户响应

不要在需要数字的位置使用该项。

EGL Java 运行时错误代码 VGJ0119E

VGJ0119E: %1 不是有效数字。

说明

当使用调试器时，用户尝试设置数字项的值，但是新值不是数字。

用户响应

使用数值。

EGL Java 运行时错误代码 VGJ0120E

VGJ0120E: 对于项 %2 上的子串运算符的起始下标，%1 不是有效值。

说明

起始下标不能小于 1 或小于该项的长度。

用户响应

将有效下标用于子串运算符的起始下标。

EGL Java 运行时错误代码 VGJ0121E

VGJ0121E: 对于项 %2 上的子串运算符的结束下标，%1 不是有效值。

说明

结束下标不能小于 1 或大于该项的长度。

用户响应

将有效下标用于子串运算符的结束下标。

EGL Java 运行时错误代码 VGJ0122E

VGJ0122E: 项 %1 上的子串运算符的结束下标为 %2，它不能小于起始下标，起始下标为 %3。

说明

子串运算符的结束下标不能小于起始下标。

用户响应

确保起始下标小于或等于结束下标。

EGL Java 运行时错误代码 VGJ0123E

VGJ0123E: 子串运算符失败: %1 不能用作字符串值。

说明

该变量不支持子串运算符。

用户响应

更改程序以使它不会对该变量使用子串运算符。

EGL Java 运行时错误代码 VGJ0124E

VGJ0124E: 不能将 %1 赋予记录。记录为 %2。

说明

数据项的类型不允许从记录赋值。

用户响应

更改程序以使它不会对数据项指定记录。

EGL Java 运行时错误代码 VGJ0125E

VGJ0125E: %1 不能用作字段。

说明

该变量不是字段。

用户响应

更改程序以使它不会将变量用作字段。

EGL Java 运行时错误代码 VGJ0126E

VGJ0126E: %1 与 %2 的比较中存在类型不兼容的问题。

说明

比较中的值的类型不兼容。

用户响应

确保比较使用兼容类型。

EGL Java 运行时错误代码 VGJ0127E

VGJ0127E: 不能对日期或时间值指定 %1。该值为 %2。

说明

不能对该项指定日期或时间值。

用户响应

不要对该项指定日期或时间值。

EGL Java 运行时错误代码 VGJ0140E

VGJ0140E: 由于尝试将数组 %2 扩展为超出它的最大大小，所以数组函数 %1 失败。

说明

该数组不能存放任何更多的值。

用户响应

修改程序，在尝试对数组进行添加之前检查它的大小。

EGL Java 运行时错误代码 VGJ0141E

VGJ0141E: 对于数组 %2，%1 不是有效的下标。当前大小: %3。最大大小: %4

说明

下标超出数组的范围。

用户响应

修改程序以使用有效的数组下标。

EGL Java 运行时错误代码 VGJ0142E

VGJ0142E: 无法更改数组 %1 的 `maximumSize`。期望值是 %2，但得到的是 %3。

说明

在 `call` 语句上传递了数组。被调用程序中的对应数组具有不同的 `maximumSize`。

用户响应

更改其中一个程序，以使两个程序使用具有相同 `maximumSize` 的数组。

EGL Java 运行时错误代码 VGJ0143E

VGJ0143E: 对于数组 %2, %1 不是有效的大小。

说明

在 call 语句上传递了数组。被调用程序将数组大小更改为小于零或大于 maxSize 属性值的值。

用户响应

更改程序, 以使它们使用属性 maxSize 的值。

EGL Java 运行时错误代码 VGJ0144E

VGJ0144E: 对于数组 %2, %1 失败。指定了太多个数。

说明

指定函数失败。其自变量是一个数组的大小, 该数组包含的元素太多了。

用户响应

更正该自变量。

EGL Java 运行时错误代码 VGJ0145E

VGJ0145E: 对于数组 %2, %1 失败。大小必须是数字数据项。

说明

指定函数失败。其自变量应该是数字数据项数组, 但它不是。

用户响应

更正该自变量。

EGL Java 运行时错误代码 VGJ0146E

VGJ0146E: 对于数组 %2, %1 失败。给定大小小于零。

说明

指定函数失败。其自变量是大小数组。其中一个大小小于零, 但这是不允许的。

用户响应

更正该自变量。

EGL Java 运行时错误代码 VGJ0147E

VGJ0147E: 对于数组 %2, %1 失败。给定最大大小小于当前大小。

说明

数组的最大大小不能更改为小于其当前大小的值。

用户响应

更正该自变量。

EGL Java 运行时错误代码 VGJ0160E

VGJ0160E: 数学函数 %1 失败，错误代码为 8 (域错误)。

说明

函数的自变量无效。

用户响应

执行下列操作:

- 根据函数的文档来更改程序逻辑，以确保函数的自变量有效；或者
- 在 try 语句中调用该函数，或者在调用该函数之前将 `VGVar.handleSysLibraryErrors` 设置为 1，以使程序能够处理该错误。

EGL Java 运行时错误代码 VGJ0161E

VGJ0161E: 数学函数 %1 失败，错误代码为 8 (域错误)。

说明

自变量必须介于 -1 与 1 之间。

用户响应

执行下列操作:

- 更改程序逻辑以确保传递给函数的自变量介于 -1 与 1 之间；或者
- 在 try 语句中调用该函数，或者在调用该函数之前将 `VGVar.handleSysLibraryErrors` 设置为 1，以使程序能够处理该错误。

EGL Java 运行时错误代码 VGJ0162E

VGJ0162E: 数学函数 `atan2` 失败，错误代码为 8 (域错误)。

说明

两个自变量不能都是零。

用户响应

执行下列操作:

- 更改程序逻辑以确保至少有一个传递给函数的自变量不为零；或者

- 在 try 语句中调用该函数，或者在调用该函数之前将 `VGVar.handleSysLibraryErrors` 设置为 1，以使程序能够处理该错误。

EGL Java 运行时错误代码 VGJ0163E

VGJ0163E: 数学函数 %1 失败，错误代码为 8 (域错误)。

说明

第二个自变量一定不能为零。

用户响应

执行下列操作:

- 更改程序逻辑以确保第二个自变量不为零；或者
- 在 try 语句中调用该函数，或者在调用该函数之前将 `VGVar.handleSysLibraryErrors` 设置为 1，以使程序能够处理该错误。

EGL Java 运行时错误代码 VGJ0164E

VGJ0164E: 数学函数 %1 失败，错误代码为 8 (域错误)。

说明

自变量必须大于零。

用户响应

执行下列操作:

- 更改程序逻辑以确保传递给函数的自变量大于零；或者
- 在 try 语句中调用该函数，或者在调用该函数之前将 `VGVar.handleSysLibraryErrors` 设置为 1，以使程序能够处理该错误。

EGL Java 运行时错误代码 VGJ0165E

VGJ0165E: 数学函数 `pow` 失败，错误代码为 8 (域错误)。

说明

如果第一个自变量为零，则第二个自变量必须大于零。

用户响应

执行下列操作:

- 更改程序逻辑以确保如果传递给函数的第一个自变量为零，则第二个自变量大于零；或者
- 在 try 语句中调用该函数，或者在调用该函数之前将 `VGVar.handleSysLibraryErrors` 设置为 1，以使程序能够处理该错误。

EGL Java 运行时错误代码 VGJ0166E

VGJ0166E: 数学函数 `pow` 失败, 错误代码为 **8** (域错误)。

说明

如果第一个自变量小于零, 则第二个自变量必须是一个整数。

用户响应

执行下列操作:

- 更改程序逻辑以确保如果传递给函数的第一个自变量小于零, 则第二个自变量为整数; 或者
- 在 `try` 语句中调用该函数, 或者在调用该函数之前将 `VGVar.handleSysLibraryErrors` 设置为 `1`, 以使程序能够处理该错误。

EGL Java 运行时错误代码 VGJ0167E

VGJ0167E: 数学函数 `sqrt` 失败, 错误代码为 **8** (域错误)。

说明

自变量必须大于等于零。

用户响应

执行下列操作:

- 更改程序逻辑以确保传递给函数的自变量大于等于零; 或者
- 在 `try` 语句中调用该函数, 或者在调用该函数之前将 `VGVar.handleSysLibraryErrors` 设置为 `1`, 以使程序能够处理该错误。

EGL Java 运行时错误代码 VGJ0168E

VGJ0168E: 数学函数 `%1` 失败, 错误代码为 **12** (范围错误)。

说明

不能将中间结果或最终结果表示为双精度浮点数或者具有结果项的精度。

用户响应

执行下列操作:

- 更改程序逻辑以确保目标项的大小足以容纳结果值; 或者
- 更改程序逻辑以便函数的自变量具有不会导致此问题的值; 或者
- 在 `try` 语句中调用该函数, 或者在调用该函数之前将 `VGVar.handleSysLibraryErrors` 设置为 `1`, 以使程序能够处理该错误。

EGL Java 运行时错误代码 VGJ0200E

VGJ0200E: 字符串函数 `%1` 失败, 错误代码为 **8**。

说明

下标必须介于 1 与字符串长度之间。

用户响应

执行下列操作:

- 更改程序逻辑以确保与下标相关的函数自变量的范围介于 1 与字符串长度之间; 或者
- 在 try 语句中调用该函数, 或者在调用该函数之前将 `VGVar.handleSysLibraryErrors` 设置为 1, 以使程序能够处理该错误。

EGL Java 运行时错误代码 VGJ0201E

VGJ0201E: 字符串函数 `%1` 失败, 错误代码为 12。

说明

长度必须大于零。

用户响应

执行下列操作:

- 更改程序逻辑以确保传递给函数的长度自变量的值大于零; 或者
- 在 try 语句中调用该函数, 或者在调用该函数之前将 `VGVar.handleSysLibraryErrors` 设置为 1, 以使程序能够处理该错误。

EGL Java 运行时错误代码 VGJ0202E

VGJ0202E: 字符串函数 `setNullTerminator` 失败, 错误代码为 16。

说明

目标字符串的最后一个字节必须为空白或空字符。

用户响应

执行下列操作:

- 更改程序逻辑以确保目标字符串的最后一个字节为空白或空字符; 或者
- 在 try 语句中调用该函数, 或者在调用该函数之前将 `VGVar.handleSysLibraryErrors` 设置为 1, 以使程序能够处理该错误。

EGL Java 运行时错误代码 VGJ0203E

VGJ0203E: 字符串函数 `%1` 失败, 错误代码为 20。

说明

DBCHAR 或 UNICODE 子串的下标必须为奇数, 以便下标标识字符的第一个字节。

用户响应

执行下列操作:

- 更改程序逻辑以确保传递给函数的下标自变量有效; 或者
- 在 try 语句中调用该函数, 或者在调用该函数之前将 `VGVar.handleSysLibraryErrors` 设置为 1, 以使程序能够处理该错误。

EGL Java 运行时错误代码 VGJ0204E

VGJ0204E: 字符串函数 %1 失败, 错误代码为 24。

说明

DBCHAR 或 UNICODE 子串的长度必须为偶数, 以指示整个字符数。

用户响应

执行下列操作:

- 更改程序逻辑以确保传递给函数的长度自变量具有有效值; 或者
- 在 try 语句中调用该函数, 或者在调用该函数之前将 `VGVar.handleSysLibraryErrors` 设置为 1, 以使程序能够处理该错误。

EGL Java 运行时错误代码 VGJ0215E

VGJ0215E: %1 被传递了非数字字符串 %2。

说明

长度自变量定义的字符串部分中的每个字符都必须是数字。

用户响应

更改程序逻辑, 以使长度自变量定义的字符串部分中的每个字符都是数字。

EGL Java 运行时错误代码 VGJ0216E

VGJ0216E: %1 不是 %2 的有效日期掩码。

说明

在属性文件中定义的用于与函数配合使用的日期掩码无效。

日期掩码的有效字符如下所示:

D、M 和 Y

D 表示日, M 表示月, Y 表示年

分隔符 除了 D、M 或 Y 之外的任何非数字的单字节字符集。

有效日期掩码可以采用下列任何格式:

- 长格里历

格里历掩码的长型版本必须包含下列部分，各部分可以采用任意顺序：

YYYY 4 位数的年份

MM 2 位数的月份

DD 2 位数的日

必须用除了 D、M 或 Y 之外的任何非数字的单字节字符集来分隔各个掩码部分。

例如，使用掩码 YYYY/MM/DD 来将 1997 年 8 月 25 日显示为 1997/08/25。

- 长儒略历

儒略历掩码的长型版本必须包含下列部分，各部分可以采用任意顺序：

YYYY 4 位数的年份

DDD 3 位数的天数

必须用除了 D、M 或 Y 之外的任何非数字的单字节字符集来分隔各个掩码部分。

例如，可以使用掩码 DDD-YYYY 来将 1997 年 8 月 25 日显示为 237-1997。

用户响应

将日期掩码属性更改为有效值，并重新启动程序。如果没有定义任何日期掩码属性，则使用缺省日期掩码。

可以使用属性 `vgj.datemask.gregorian.long.NNN` 和 `vgj.datemask.julian.long.NNN` 来设置日期掩码，其中，`NNN` 是当前的 NLS 代码。

EGL Java 运行时错误代码 VGJ0217E

VGJ0217E: 带有自变量 %1 的转换函数发生错误: %2

说明

转换自变量数据的尝试失败。故障原因已包括在消息中。

用户响应

使用错误消息来诊断和更正问题。

EGL Java 运行时错误代码 VGJ0218E

VGJ0218E: GetMessage 失败。找不到键 %1 的消息。

说明

找不到传递至 `getMessage` 系统函数的键的消息。

用户响应

添加该消息，或者使用另一个键。

EGL Java 运行时错误代码 VGJ0250E

VGJ0250E: 未能从包含部件 %2 中检索 %1 项。

说明

发生了内部错误。已尝试访问记录或表中具有指定索引的项。

用户响应

执行下列操作:

1. 记录消息号和消息文本。

注: 错误消息包括下列重要信息:

- 错误的发生位置
 - 内部错误的类型
2. 记录发生此消息的情况。
 3. 有关如何向“IBM 支持中心”报告可能存在的缺陷的更多指示信息, 参阅 *EGL Installation Guide*。

EGL Java 运行时错误代码 VGJ0300E

VGJ0300E: 未能装入表 %1 的表文件。找不到名为 %2 或 %3 的文件。

说明

在任何资源位置都找不到指定的文件。搜索了所有资源位置来查找第一个文件。如果不存在这样的文件, 则搜索所有资源位置来查找第二个文件。

根据用来查找表文件的机制的不同, 资源位置也会有所变化。

如果在 applet 中遇到了错误, 则资源位置是指服务器上的位置, 并且会根据 Java 虚拟机实现的不同而有所变化。然而, 所有实现都应该搜索服务器上由 CODEBASE 值指定的目录。此值是由包含 applet 的 HTML 文件中的 APPLET 标记设置的。如果未指定 CODEBASE 值, 则它将缺省为 Web 服务器上包含 HTML 文件的目录。

如果在应用程序中遇到了错误, 则有效的资源位置如下所示:

- 启动 Java 虚拟机所在的目录 (可执行文件的工作目录)。
- 在 CLASSPATH 中为正在运行的应用程序指定的任何目录。此值的规范依赖于系统。在某些系统上, 可以将它作为环境变量指定。所有系统都允许在调用 Java 虚拟机时使用 -classpath 选项来指定该目录。有关 CLASSPATH 值的更多信息, 请参阅随 Java 虚拟机的副本一起提供的文档。

用户响应

首先, 找到表文件, 并确保设置了访问它所需要的许可权。

如果在 applet 中发生了错误, 或者在应用程序中发生了错误, 而您又不想修改资源位置的现有设置, 则将表文件复制到有效的资源位置。

否则, 完成下列其中一项指示信息:

- 如果 Java 解释器将使用 CLASSPATH 环境变量的值，则将包含表文件的目录添加到 CLASSPATH 的当前值中。
- 调用 Java 解释器时，使用 -classpath 选项来指定包含表文件的目录。如果指定 -classpath 选项将覆盖 CLASSPATH 环境变量的值，则除了作为资源位置添加的任何目录之外，还需要指定 Java 运行时类（例如 classes.zip 或 rt.jar）的路径。

如果启用了程序跟踪，则可能还会提供其它诊断信息。

EGL Java 运行时错误代码 VGJ0301E

VGJ0301E: 未能装入表 %2 的表文件 %1，原因是在对表头执行读操作期间返回了不正确的字节数。

说明

存在下列其中一种情况：

- 表文件已损坏。
- 该表文件不是使用 EGL 或 VisualAge Generator 生成的。

用户响应

重新生成该表。

如果启用了程序跟踪，则可能还会提供其它诊断信息。

如果问题仍然存在，则执行以下操作：

1. 记录消息号和消息文本。

注： 错误消息包括下列重要信息：

- 错误的发生位置
 - 内部错误的类型
2. 记录发生此消息的情况。
 3. 有关如何向“IBM 支持中心”报告可能存在的缺陷的更多指示信息，参阅 *EGL Installation Guide*。

EGL Java 运行时错误代码 VGJ0302E

VGJ0302E: 未能装入表 %2 的表文件 %1，原因是在检查表头期间遇到了意外的幻数。

说明

存在下列其中一种情况：

- 表文件已损坏。
- 该表文件不是使用 EGL 或 VisualAge Generator 生成的。

用户响应

重新生成该表。

如果启用了程序跟踪，则可能还会提供其它诊断信息。

如果问题仍然存在，则执行以下操作：

1. 记录消息号和消息文本。

注： 错误消息包括下列重要信息：

- 错误的发生位置
- 内部错误的类型

2. 记录发生此消息的情况。

3. 有关如何向“IBM 支持中心”报告可能存在的缺陷的更多指示信息，参阅 *EGL Installation Guide*。

EGL Java 运行时错误代码 VGJ0303E

VGJ0303E: 未能装入表 %2 的表文件 %1，原因是在执行读取或关闭操作期间发生了内部 I/O 错误。

说明

存在下列其中一种情况：

- 表文件已损坏。
- 该表文件不是使用 EGL 或 VisualAge Generator 生成的。

用户响应

重新生成该表。

如果启用了程序跟踪，则可能还会提供其它诊断信息。

如果问题仍然存在，则执行以下操作：

1. 记录消息号和消息文本。

注： 错误消息包括下列重要信息：

- 错误的发生位置
- 内部错误的类型

2. 记录发生此消息的情况。

3. 有关如何向“IBM 支持中心”报告可能存在的缺陷的更多指示信息，参阅 *EGL Installation Guide*。

EGL Java 运行时错误代码 VGJ0304E

VGJ0304E: 未能装入表 %2 的表文件 %1，原因是在对表数据执行读操作期间返回了不正确的字节数。

说明

存在下列其中一种情况:

- 在更改了表文件的列之后重新生成了表文件, 但是未重新生成尝试装入该表的程序。在更改列定义之后只生成表将导致表文件中的定义与表类文件中的定义(仅在运行时代码生成期间才会生成此定义)之间存在不一致。
- 表文件已损坏。
- 该表文件不是使用 EGL 或 VisualAge Generator 生成的。

用户响应

执行下列操作:

- 如果未更改列定义, 则重新生成该表。
- 如果已更改列定义, 则除去更改并重新生成该表, 或者重新生成使用该表的程序的运行时代码。

如果启用了程序跟踪, 则可能还会提供其它诊断信息。

如果问题仍然存在, 则执行以下操作:

1. 记录消息号和消息文本。

注: 错误消息包括下列重要信息:

- 错误的发生位置
 - 内部错误的类型
2. 记录发生此消息的情况。
 3. 有关如何向“IBM 支持中心”报告可能存在的缺陷的更多指示信息, 参阅 *EGL Installation Guide*。

EGL Java 运行时错误代码 VGJ0305E

VGJ0305E: 未能装入表 %2 的表文件 %1。在表文件中遇到的项 %3 的数据的格式不正确。相应的数据格式错误是: %4

说明

存在下列其中一种情况:

- 在更改了表文件的列之后重新生成了表文件, 但是未重新生成尝试装入该表的 applet 或应用程序。在更改列定义之后只生成表将导致表文件中的定义与表类文件中的定义(仅在运行时代码生成期间才会生成此定义)之间存在不一致。
- 表文件已损坏。
- 表文件不是使用 Rational Application Developer for z/OS 或 VisualAge Generator 生成的。

用户响应

执行下列操作:

- 如果未更改列定义, 则重新生成该表。

- 如果已更改列定义，则除去更改并重新生成该表，或者重新生成使用该表的程序的运行时代码。

如果问题仍然存在，则执行以下操作：

1. 记录消息号和消息文本。

注： 错误消息包括下列重要信息：

- 错误的发生位置
- 内部错误的类型

2. 记录发生此消息的情况。
3. 有关如何向“IBM 支持中心”报告可能存在的缺陷的更多指示信息，参阅 *EGL Installation Guide*。

EGL Java 运行时错误代码 VGJ0306E

VGJ0306E: 未能装入表 %2 的表文件 %1，原因是该表文件中的数据适用于的表类型与表 %2 的表类型不同。

说明

存在下列其中一种情况：

- 在更改了表文件的列之后重新生成了表文件，但是未重新生成尝试装入该表的 applet 或应用程序。在更改列定义之后只生成表将导致表文件中的定义与表类文件中的定义（仅在运行时代码生成期间才会生成此定义）之间存在不一致。
- 表文件已损坏。
- 该表文件不是使用 EGL 或 VisualAge Generator 生成的。

用户响应

执行下列操作：

- 如果未更改表类型，则重新生成该表。
- 如果已更改表类型，则编辑表定义以使其具有正确类型并重新生成表，或者重新生成使用该表的程序的运行时代码。

如果启用了程序跟踪，则可能还会提供其它诊断信息。

如果问题仍然存在，则执行以下操作：

1. 记录消息号和消息文本。

注： 错误消息包括下列重要信息：

- 错误的发生位置
- 内部错误的类型

2. 记录发生此消息的情况。
3. 有关如何向“IBM 支持中心”报告可能存在的缺陷的更多指示信息，参阅 *EGL Installation Guide*。

EGL Java 运行时错误代码 VGJ0307E

VGJ0307E: 由于表文件 %1 是 VisualAge Generator C++ 表文件, 并且不具有大尾数法格式, 所以未能装入表 %2 的表文件 %1。

说明

如果用来对表中的数字数据进行编码的字节定序方法是大尾数法的话, 由 VisualAge Generator C++ 生成器生成的表文件就只能与 Java 程序配合使用。

用户响应

以大尾数法格式重新生成该表, 或者将该表作为与平台无关的 Java 表重新生成。

要以大尾数法格式重新生成该表, 请使用 VisualAge Generator 来为采用大尾数法的 C++ 目标系统 (例如 AIX) 生成该表。要将表重新生成成为独立于 Java 平台的表, 使用 VisualAge Generator 或 EGL 来对 Java 目标系统生成该表。

如果启用了程序跟踪, 则可能还会提供其它诊断信息。

EGL Java 运行时错误代码 VGJ0308E

VGJ0308E: 未能装入表 %2 的表文件 %1。表文件 %1 是 VisualAge Generator C++ 表文件, 并且在表 (%3) 中使用的字符编码在运行时系统上不受支持。

说明

仅当用于表数据的字符编码类型与运行时系统使用的编码类型相同时, 由 VisualAge Generator C++ 生成器生成的表文件才能与 Java 程序配合使用。

用户响应

执行下列操作:

1. 确定在系统上使用的字符编码。Java 程序使用 ASCII 或 EBCDIC 字符编码。大多数工作站使用 ASCII 编码。大多数主机平台使用 EBCDIC 编码。如果您不知道系统上使用的编码, 请与系统管理员联系。
2. 使用正确的字符编码重新生成该表, 或者将该表作为与平台无关的 Java 表重新生成。

要使用正确的字符编码来重新生成表, 使用 VisualAge Generator 来为目标系统或使用相同字符编码的另一个 C++ 目标系统生成表。要将表重新生成成为独立于 Java 平台的表, 使用 VisualAge Generator 或 EGL 来对 Java 目标系统生成该表。

如果启用了程序跟踪, 则可能还会提供其它诊断信息。

EGL Java 运行时错误代码 VGJ0315E

VGJ0315E: 在执行表卸装过程中找不到表 %1 的共享表条目。

说明

发生了内部错误。

用户响应

执行下列操作:

1. 记录消息号和消息文本。

注: 错误消息包括下列重要信息:

- 错误的发生位置
- 内部错误的类型

2. 记录发生此消息的情况。
3. 有关如何向“IBM 支持中心”报告可能存在的缺陷的更多指示信息, 参阅 *EGL Installation Guide*。

EGL Java 运行时错误代码 VGJ0320E

VGJ0320E: 对表 %1 执行的编辑例程在将表列 %2 与字段 %3 作比较时失败。

说明

该表列和该字段所具有的类型对比较而言是无效的。

用户响应

执行下列其中一项操作:

- 通过执行下列操作, 确保列和字段的类型对于比较而言是有效的:
 1. 更正列的类型或字段的类型, 以使比较将是有效的。
 2. 重新生成程序。
 3. 运行程序。
- 修改程序, 对编辑例程使用另一个表, 以使列与字段的比较将是有效的。

有关更多信息, 参阅跟踪输出。

EGL Java 运行时错误代码 VGJ0330E

VGJ0330E: 在消息表 %2 中找不到具有标识 %1 的消息。

说明

在下列操作中可能会发生此错误:

- 查找表单的 msgField 值。
- 查找具有作为编辑消息指定的标识的值。

存在下列其中一种情况:

- 在消息表中不存在具有此标识的消息。
- 该表的表文件或消息资源束已损坏。

用户响应

执行下列其中一项操作:

- 通过执行下列操作，确保存在具有该消息标识的消息：
 1. 如果不存在该消息，则将具有该消息标识的消息添加到表中。
 2. 重新生成该表。
 3. 运行程序。
- 修改程序以使用另一条在表中已定义的消息。
- 修改程序以使用另一个消息表，该消息表包含具有该消息标识的消息。

EGL Java 运行时错误代码 VGJ0331E

VGJ0331E: 未能装入消息表文件 %1。

说明

未能装入程序的消息表的类，或者未能创建该类的实例。

用户响应

确保已生成该消息表。

EGL Java 运行时错误代码 VGJ0350E

VGJ0350E: 调用程序 %1 时发生了错误。错误代码为 %2。

说明

对指定的程序进行的远程或 EJB 调用失败。

用户响应

如果启用了程序跟踪，则可能还会提供其它诊断信息。

EGL Java 运行时错误代码 VGJ0351E

VGJ0351E: 落实失败: %1

说明

未能落实资源。

用户响应

如果启用了程序跟踪，则可能还会提供其它诊断信息。

EGL Java 运行时错误代码 VGJ0352E

VGJ0352E: 回滚失败: %1

说明

未能回滚资源。

用户响应

如果启用了程序跟踪，则可能还会提供其它诊断信息。

EGL Java 运行时错误代码 VGJ0400E

VGJ0400E: 对函数 %2 使用了无效的参数索引 %1。

说明

这是内部错误。

用户响应

与 IBM 支持机构联系。

EGL Java 运行时错误代码 VGJ0401E

VGJ0401E: 对函数 %1 参数 %2 检测到无效的参数描述符。

说明

这是内部错误。

用户响应

与 IBM 支持机构联系。

EGL Java 运行时错误代码 VGJ0402E

VGJ0402E: 用于函数或程序 %2 的参数 %1 的值的类型无效。

说明

由于该值的类型与参数的类型不兼容，所以该值不能作为参数传递。

用户响应

执行下列其中一项操作：

- 更改参数的定义以与该值的类型匹配。
- 更改该值的类型以与参数的定义匹配。

EGL Java 运行时错误代码 VGJ0403E

VGJ0403E: 在运行脚本 %1 时发生错误。异常文本是 %2。

说明

该脚本导致抛出了异常。

用户响应

更正程序逻辑以避免发生错误。

EGL Java 运行时错误代码 VGJ0416E

VGJ0416E: 调用程序 %1 时发生了错误。错误代码为 %2 (%3)。

说明

尝试运行被调用程序时抛出了异常。该问题可能是由于下列其中一种情况引起的:

- 程序可能没有创建新进程的许可权。
- 被调用程序可能不存在。
- 在系统路径中可能找不到被调用程序。

用户响应

执行下列操作:

1. 验证程序是否具有创建新进程的许可权。
2. 验证被调用程序是否存在。
3. 验证在系统路径中是否能够找到被调用程序。

如果问题仍然存在, 则执行以下操作:

1. 记录消息号和消息文本。

注: 错误消息包括下列重要信息:

- 错误的发生位置
 - 内部错误的类型
2. 记录发生此消息的情况。
 3. 有关如何向“IBM 支持中心”报告可能存在的缺陷的更多指示信息, 参阅 *EGL Installation Guide*。

EGL Java 运行时错误代码 VGJ0450E

VGJ0450E: 对 I/O 对象 %2 的 I/O 操作 %1 由于以下原因而失败: %3。

说明

在 try 语句外部, 或者当系统变量 `sysVar.handleHardIoErrors` 的值为零时, EGL I/O 语句失败。

用户响应

查看错误消息并适当地作出响应。

EGL Java 运行时错误代码 VGJ0500E

VGJ0500E: 必需的字段未接收到输入 - 重新输入。

说明

未在字段中输入数据。该字段被定义为是必需的。

用户响应

在字段中输入数据，或按绕过编辑键以绕过编辑检查。空格不能满足任何类型的字段的数据输入需求。另外，零不能满足数字字段的数据输入需求。程序继续。

EGL Java 运行时错误代码 VGJ0502E

VGJ0502E: 输入数据类型错误 - 重新输入。

说明

字段中的数据不是有效的数字数据。该字段被定义为数字字段。

用户响应

仅在此字段中输入数字数据，或者按绕过编辑键以绕过编辑检查。在任何一种情况下，程序都将继续。

EGL Java 运行时错误代码 VGJ0503E

VGJ0503E: 超出了允许的有效位数 - 重新输入。

说明

已将数据输入到数字字段中，该字段是使用小数位、符号、货币符号或数字分隔符编辑定义的。输入数据超出了可以在编辑条件下显示的有效位数。输入的数字太大。有效位数不能超出字段长度减去小数位数再减去编辑字符所需的位数。

用户响应

输入具有更少有效位数的数字。

EGL Java 运行时错误代码 VGJ0504E

VGJ0504E: 输入不在已定义的范围之内 - 重新输入。

说明

字段中的数据不在为此项定义的有效数据的范围之内。

用户响应

输入位于已定义范围内的数据，或按绕过编辑键以绕过编辑检查。在任何一种情况下，程序都将继续。

EGL Java 运行时错误代码 VGJ0505E

VGJ0505E: 输入最小长度错误 - 重新输入。

说明

字段中的数据未包含足够的字符来满足所要求的最小长度。

用户响应

输入满足最小长度所需的字符数目，或按绕过编辑键以绕过编辑检查。在任何一种情况下，程序都将继续。

EGL Java 运行时错误代码 VGJ0506E

VGJ0506E: 表编辑有效性错误 - 重新输入。

说明

字段中的数据不符合对变量字段定义的表编辑需求。

用户响应

输入符合表编辑需求的数据，或按绕过编辑键以绕过编辑检查。在任何一种情况下，程序都将继续。

EGL Java 运行时错误代码 VGJ0507E

VGJ0507E: 输入模数检查错误 - 重新输入。

说明

字段中的数据不符合对变量字段定义的模数检查需求。

用户响应

输入符合对变量字段定义的模数检查的数据，或按绕过编辑键以绕过编辑检查。在任何一种情况下，程序都将继续。

EGL Java 运行时错误代码 VGJ0508E

VGJ0508E: 输入对已定义的时间或日期格式 %1 无效。

说明

字段中的数据（用日期编辑定义）不符合格式规范需求。

用户响应

按照消息中显示的正确格式来输入日期。

EGL Java 运行时错误代码 VGJ0510E

VGJ0510E: 输入对布尔字段无效。

说明

在字段中输入的值不符合布尔检查。输入到布尔字段中的内容对于字符字段必须是“Y”或“N”，对于数字字段必须是 1 或 0。

用户响应

对于字符字段，输入“Y”或“N”，对于数字字段，输入 1 或 0，或者按绕过编辑键以绕过编辑检查。在任何一种情况下，程序都将继续。

EGL Java 运行时错误代码 VGJ0511E

VGJ0511E: 未对 %2 定义编辑表 %1。

说明

已请求用户消息，但未对该程序定义用户消息表前缀。

用户响应

请程序开发者执行下列其中一项操作：

- 将消息表前缀添加到程序规范中，并重新生成该程序。
- 从字段编辑中除去该用户消息号，并重新生成。

EGL Java 运行时错误代码 VGJ0512E

VGJ0512E: 十六进制数据无效。

说明

变量字段中的数据必须具有十六进制格式。输入的一个或多个字符未出现在以下集合中： a b c d e f A B C D E F 0 1 2 3 4 5 6 7 8 9

用户响应

在变量字段中仅输入十六进制字符。这些字符是向左对齐的，并且用字符 0 进行填充。不允许嵌入空格。

EGL Java 运行时错误代码 VGJ1234E

VGJ1234E: 输入的值无效，这是因为它与已设置的模式不匹配。

说明

输入的值与模式不匹配

用户响应

按模式中指定的那样输入值。

EGL Java 运行时错误代码 VGJ1234E

VGJ0514E: 输入最大长度错误 - 重新输入。

说明

超出对此字段设置的指定最大长度。

用户响应

不要超出指定的最大长度。

EGL Java 运行时错误代码 VGJ0516E

VGJ0516E: 输入不在定义的列表中 - 重新输入。

说明

输入不在定义的列表中 - 重新输入。

用户响应

输入不在定义的列表中 - 重新输入。

EGL Java 运行时错误代码 VGJ0517E

VGJ0517E: 指定的日期 / 时间格式 %1 无效。

说明

指定的日期 / 时间格式无效。

用户响应

指定的日期 / 时间格式无效。

EGL Java 运行时错误代码 VGJ0600E

VGJ0600E: 无法获取程序 %1 的链接。

说明

由于下列其中一项原因，在 CSO 属性文件中找不到指定的程序的条目：

- 在 GatewayServlet 配置中指定了不正确的属性文件。
- 未在 CSO 属性文件中指定程序的条目。
- CSO 属性文件不在 GatewayServlet 配置所指定的目录中。

用户响应

与 Web 服务器管理员联系以确保执行了下列操作：

- 确保 GatewayServlet 配置使用 linkageTable 初始化参数指定了正确的 CSO 属性文件。

- 确保在 CSO 属性文件中定义了程序。

EGL Java 运行时错误代码 VGJ0601E

VGJ0601E: 在尝试调用入口点程序 %1 时发生异常。异常: %2。消息: %3。

说明

在尝试调用入口点程序时发生了未说明的错误。异常和消息将进一步定义错误。入口点页或程序向用户提供可以使用 GatewayServlet 启动的程序的菜单。

用户响应

与 Web 服务器管理员联系以确保在 GatewayServlet 配置中正确地指定了入口点页或入口程序。

EGL Java 运行时错误代码 VGJ0603E

VGJ0603E: bean %1 无效。

说明

页 Bean 或 bean 名称无效。

用户响应

与 Web 服务器管理员联系以确保 bean 名称正确，并且确保已部署页 Bean 和 Java Server Page 并使它们可供 GatewayServlet 使用。

EGL Java 运行时错误代码 VGJ0604E

VGJ0604E: 在尝试装入 bean %1 时发生异常。异常: %2。消息: %3。

说明

在尝试装入页 Bean 时发生未说明的错误。异常和消息将进一步定义错误。

用户响应

与 Web 服务器管理员联系以确保 bean 名称正确，并且确保已部署页 Bean 和 Java Server Page 并使它们可供 GatewayServlet 使用。

EGL Java 运行时错误代码 VGJ0607E

VGJ0607E: 在服务器 %1 与 bean %2 之间发生版本不匹配。

说明

用户界面记录 Bean 的版本与服务器程序使用的用户界面记录的版本不匹配。要进行正确的操作，版本必须兼容。

用户响应

与程序开发者联系，并生成程序和用户界面记录 bean。与 Web 服务器管理员联系以确保将用户界面记录 bean 部署到正确的位置。

EGL Java 运行时错误代码 VGJ0608E

VGJ0608E: 尝试在 bean %1 中设置数据时发生错误。异常: %2。消息: %3。

说明

尝试将服务器应用程序提供的记录数据设置到用户界面记录 Bean 中时发生异常。包括了异常和消息，它们可以帮助您确定问题。

用户响应

使用包括在消息中的异常和消息来进行问题确定。

EGL Java 运行时错误代码 VGJ0609I

VGJ0609I: 正在为用户 %1 绑定网关会话。

说明

此参考消息出现在应用程序服务器的标准输出或标准错误中。每当为用户创建 Web 会话时，此消息就会出现。

用户响应

不需要进行响应。

EGL Java 运行时错误代码 VGJ0610I

VGJ0610I: 正在为用户 %1 取消绑定网关会话。

说明

此参考消息出现在应用程序服务器的标准输出或标准错误中。每当为用户结束 Web 会话时，此消息就会出现。在一段时间不活动之后，或者如果发生导致会话终止的严重错误，则会话将结束。

用户响应

不需要进行响应。

EGL Java 运行时错误代码 VGJ0611E

VGJ0611E: 无法建立与 SessionIDManager 的连接。

说明

GatewayServlet 无法连接至 SessionIDManager。SessionIDManager 是为网关用户提供会话标识的组件。将为每个活动会话获取一个会话标识，服务器程序使用该会话标识来保存和恢复应用程序数据。

SessionIDManager 是一个独立的应用程序，它侦听该标识的连接和请求。当会话结束时，SessionIDManager 将使该会话标识可供其它会话使用。要运行 GatewayServlet，SessionIDManager 必须处于活动状态。

用户响应

与 Web 服务器管理员联系以启动 SessionIDManager。如果已启动，则必须在 GatewayServlet 的配置中设置 SessionIDManager 的位置。

EGL Java 运行时错误代码 VGJ0612I

VGJ0612I: 已经为用户 %1 将网关会话连接至 SessionIDManager。

说明

此参考消息出现在 Web 服务器的标准输出或标准错误中。为了获取会话标识，已将会话成功地连接至 SessionIDManager。服务器程序使用该会话标识来保存和恢复程序数据。

用户响应

不需要进行响应。

EGL Java 运行时错误代码 VGJ0614E

VGJ0614E: 在 GatewayServlet 配置中缺少必需参数 %1。

说明

在 servlet 配置中未指定必需参数。没有这些参数，GatewayServlet 无法运行。

用户响应

与 Web 服务器管理员联系以确保正确地配置了 GatewayServlet。参考应用程序服务器文档以确定如何配置 servlet 参数。

EGL Java 运行时错误代码 VGJ0615E

VGJ0615E: 不允许在这个 EGL Action Invoker 实例上运行 Web 事务 %1。

说明

创建或检索程序的 GatewayRequestHandler 时出现问题。

用户响应

确保已生成命名的应用程序并已将其部署至服务器。

EGL Java 运行时错误代码 VGJ0616E

VGJ0616E: 网关参数 %1 未指定有效的类: %2

说明

指定的网关属性中标识的类无法被装入或实例化。

用户响应

确保已将命名的类部署至服务器并在网关属性文件中正确地指定了该类。

EGL Java 运行时错误代码 VGJ0617E

VGJ0617E: 请在网关属性文件中提供有效的公用用户信息。

说明

在网关属性文件中指定的公用用户名或密码无效。

用户响应

确保网关属性文件中指定的公用用户名和密码值是正确的。

EGL Java 运行时错误代码 VGJ0700E

VGJ0700E: 在数据库连接期间发生了错误: %1。

说明

在尝试连接至数据库时发生了错误。错误消息以来自数据库管理系统的文本结束。

用户响应

查看错误消息并适当地作出响应。如果启用了程序跟踪，则可能还会提供其它诊断信息。

EGL Java 运行时错误代码 VGJ0701E

VGJ0701E: 必须在执行 SQL I/O 操作之前建立数据库连接。

说明

在建立数据库连接之前尝试了 SQL I/O 操作。

用户响应

仅当程序创建了数据库连接之后 SQL I/O 操作才有效。程序可以根据程序属性创建缺省连接，并且可以通过运行 connect 系统函数来覆盖缺省值。查看 EGL 帮助页面以获取有关程序属性和设置数据库访问的详细信息。

EGL Java 运行时错误代码 VGJ0702E

VGJ0702E: 在执行 SQL I/O 操作 %1 期间发生了错误: %2。

说明

在执行指定的 SQL I/O 操作期间发生了错误。消息以来自于数据库管理系统的文本结束。

用户响应

查看消息并适当地作出响应。

如果启用了程序跟踪, 则可能还会提供其它诊断信息。

EGL Java 运行时错误代码 VGJ0703E

VGJ0703E: 在设置 SQL I/O 操作 %1 期间发生了错误。%2。

说明

在设置指定的 SQL I/O 操作期间发生了错误。

用户响应

查看消息并适当地作出响应。

如果启用了程序跟踪, 则可能还会提供其它诊断信息。

EGL Java 运行时错误代码 VGJ0705E

VGJ0705E: 与数据库 %1 断开连接时发生了错误。%2。

说明

尝试与指定的数据库断开连接时发生了错误。错误消息以来自数据库管理系统的文本结束。

用户响应

查看消息并适当地作出响应。

如果启用了程序跟踪, 则可能还会提供其它诊断信息。

EGL Java 运行时错误代码 VGJ0706E

VGJ0706E: 无法设置与数据库 %1 的连接。该连接不存在。

说明

尝试设置与指定的数据库的连接时发生了错误。只能将连接设置为事务中的活动数据库连接。

用户响应

确保数据库的名称与为事务建立的其中一个活动数据库连接相匹配。

如果启用了程序跟踪，则可能还会提供其它诊断信息。

EGL Java 运行时错误代码 VGJ0707E

VGJ0707E: %1 上发生了 SQL I/O 顺序错误。

说明

在下列情况下可能会发生顺序错误:

- 发生了 EGL replace 或 delete, 但是之前没有对同一个 SQL 记录执行 setupd 或 update 语句
- 发生了 EGL scan, 但是之前没有对同一个 SQL 记录执行 setupd 或 setinq 语句

消息标识了程序尝试执行的最后一个 I/O 操作, 不管是 replace、delete 还是 scan。

用户响应

确保 EGL 语句的顺序正确。

如果启用了程序跟踪, 则可能还会提供其它诊断信息。

EGL Java 运行时错误代码 VGJ0708E

VGJ0708E: 装入 JDBC 驱动程序类 %1 时出错

说明

装入 JDBC 驱动程序类时发生错误, 那些类是 SQL I/O 所必需的。

用户响应

确保在属性 vgi.jdbc.drivers 中正确指定了 JDBC 驱动程序类。如果需要多个类, 则用分号将它们名称隔开。并确保可以在类路径中某个位置找到那些类。

EGL Java 运行时错误代码 VGJ0709E

VGJ0709E: 一个语句 (%1) 使用了尚未编译的预编译语句。

说明

错误消息中指定的预编译语句不存在。预编译语句是通过调用 EGL prepare 语句创建的。

用户响应

通过在使用预编译语句之前添加 prepare 来更正程序逻辑。

EGL Java 运行时错误代码 VGJ0710E

VGJ0710E: %1 语句使用了已关闭或不存在的结果集。

说明

由于该语句使用的结果集尚未打开或不存在，所以不能使用该结果集。

用户响应

更正程序逻辑以避免使用无效的结果集。

EGL Java 运行时错误代码 VGJ0711E

VGJ0711E: 在连接至数据库 %1 时发生了错误: %2

说明

未能建立与消息中指定的数据库的连接。

用户响应

使用此消息的错误部分来诊断和更正问题。

EGL Java 运行时错误代码 VGJ0712E

VGJ0712E: 无法连接至缺省数据库。未指定缺省数据库的名称。

说明

未指定缺省数据库的名称，因此程序无法连接至该数据库。

用户响应

可以通过数种方法指定缺省数据库的名称。必须设置属性 `vgj.jdbc.default.database.programName`（其中 `programName` 是程序的名称）和 `vgj.jdbc.default.database` 的其中之一。该属性的值可以是缺省数据库的实际名，也可以是缺省数据库的逻辑名。使用逻辑名时，必须设置另一个属性：`vgj.jdbc.database.logicalName`。此属性的值必须是缺省数据库的实际名。

EGL Java 运行时错误代码 VGJ0713E

VGJ0713E: GET 失败，原因是未使用 `scroll` 打开结果集 %1。

说明

仅当 OPEN 语句未指定 SCROLL 时，才允许 GET NEXT。

用户响应

将 `scroll` 选项添加至在其中创建结果集的 `open` 语句。

EGL Java 运行时错误代码 VGJ0750E

VGJ0750E: 未能为文件 %1 创建 I/O 驱动程序。%2

说明

在为指定的文件创建 I/O 驱动程序时发生了故障。在下列情况下可能会发生此错误:

- 在对与指定的文件相关的记录执行第一个 I/O 操作时; 或者
- 在第一次对与指定的文件相关的记录访问系统变量 `resourceAssociation` 时。

消息的末尾指示了发生故障的原因。

用户响应

查看错误消息并适当地作出响应。

EGL Java 运行时错误代码 VGJ0751E

VGJ0751E: 在 Java 运行时属性 `vgj.ra.fileName.fileType` 中找不到文件 %1 的 `fileType` 属性。

说明

需要将以下运行时属性设置为有效的文件类型:

```
vgj.ra.fileName.fileType
```

fileName

在消息中指定的文件的名称。此文件名是与 EGL 记录相关联的逻辑文件名。

对于 MQ 记录, 该值为 `mq`; 对于串行记录, 该值为 `seqws`。该值的源是生成时资源关联部件; 明确地说, 是文件的关联元素的属性 `fileType`。

用户响应

执行下列操作:

- 将运行时 `fileType` 属性添加至运行时属性文件或部署描述符; 或者
- 在生成时设置 `fileType` 值并重新生成程序:
 - 在资源关联部件的特定于文件名的关联元素中, 设置属性 `fileType`
 - 在生成时使用的构建描述符中, 将选项 `genProperties` 设置为 `GLOBAL`

有关其它详细信息, 请参阅有关关联元素、有关 Java 运行时属性和有关设置环境的 EGL 帮助页面。

EGL Java 运行时错误代码 VGJ0752E

VGJ0752E: 在资源关联部件中对文件 %2 指定了无效的 `fileType` %1。

说明

需要将以下运行时属性设置为有效的文件类型:

```
vgj.ra.fileName.fileType
```

fileName

在消息中指定的文件的名称。此文件名是与 EGL 记录相关联的逻辑文件名。

对于 MQ 记录，该值为 mq；对于串行记录，该值为 seqws。该值的源是生成时资源关联部件；明确地说，是文件的关联元素的属性 **fileType**。

用户响应

执行下列操作：

- 更改运行时属性文件或部署描述符中的运行时 **fileType** 属性；或者
- 在生成时重置 **fileType** 值并重新生成程序：
 - 在资源关联部件的特定于文件名的关联元素中，更改属性 **fileType**
 - 在生成时使用的构建描述符中，将选项 **genProperties** 设置为 GLOBAL

有关其它详细信息，请参阅有关关联元素、有关 Java 运行时属性和有关设置环境的 EGL 帮助页面。

EGL Java 运行时错误代码 VGJ0754E

VGJ0754E: 记录长度项必须包含用于在项边界分割非字符数据的值。

说明

记录具有可变长度。写出记录数据时，记录长度项指示要写的字节数。除非该项是字符，否则数据的最后一个字节必须是项的最后一个字节。

用户响应

更改程序，以使记录长度项的值指向项的最后一个字节或者位于字符项之内。

EGL Java 运行时错误代码 VGJ0755E

VGJ0755E: **occursItem** 或 **lengthItem** 中的值太大。

说明

记录具有可变长度。尝试写出的字节数比记录当前包含的字节数多。

用户响应

更改程序以使 **lengthItem** 或 **occursItem** 的值位于记录大小之内。

EGL Java 运行时错误代码 VGJ0770E

VGJ0770E: 创建 **InitialContext** 或者查找 **java:comp/env** 环境时发生了错误。错误为 %1

说明

异常是从 `javax.naming.InitialContext` 的构造函数或者通过使用值“`java:comp/env`”调用查找方法抛出的。程序需要创建 `InitialContext` 对象并查找“`java:comp/env`”以便访问 J2EE 环境设置。

用户响应

使用异常的文本和 J2EE 环境的文档来更正问题。

EGL Java 运行时错误代码 VGJ0800E

VGJ0800E: 将 %1 赋值给 %2 是无效的。

说明

当使用调试器时，尝试了将系统变量设置为无效值。

用户响应

选择有效值，如系统变量的帮助页面所述。

EGL Java 运行时错误代码 VGJ0801E

VGJ0801E: 不能修改 %1，或者它不存在。

说明

当使用调试器时，尝试了设置不能被设置或不存在的系统变量的值。

用户响应

查看帮助页面以获取系统变量列表以及每个系统变量的描述。

EGL Java 运行时错误代码 VGJ0802E

VGJ0802E: 调试 %1 时出错: %2

说明

尝试调试 `PageHandler` 时发生了错误。

用户响应

使用此消息的错误部分来诊断和更正问题。

EGL Java 运行时错误代码 VGJ0901E

VGJ0901E: 日期 / 时间范围模式 (声明时间戳记 / 时间间隔项的长度和日期 / 时间组件的字符串) 无效。

说明

日期 / 时间范围模式（声明时间戳记 / 时间间隔项的长度和日期 / 时间组件的字符串）无效。

用户响应

对日期 / 时间范围模式作相应调整。

EGL Java 运行时错误代码 VGJ0902E

VGJ0902E: 日期 / 时间范围模式（声明时间戳记 / 时间间隔项的长度和日期 / 时间组件的字符串）的精度无效。

说明

日期 / 时间范围模式（声明时间戳记 / 时间间隔项的长度和日期 / 时间组件的字符串）的精度无效。

用户响应

对日期 / 时间范围模式的精度作相应调整。

EGL Java 运行时错误代码 VGJ0903E

VGJ0903E: 日期 / 时间范围模式（声明时间戳记 / 时间间隔项的长度和日期 / 时间组件的字符串）的起始代码无效。

说明

日期 / 时间范围模式（声明时间戳记 / 时间间隔项的长度和日期 / 时间组件的字符串）的起始代码无效。

用户响应

对日期 / 时间范围模式的起始代码作相应调整。

EGL Java 运行时错误代码 VGJ0904E

VGJ0904E: 日期 / 时间范围模式（声明时间戳记 / 时间间隔项的长度和日期 / 时间组件的字符串）的结束代码无效。

说明

日期 / 时间范围模式（声明时间戳记 / 时间间隔项的长度和日期 / 时间组件的字符串）的结束代码无效。

用户响应

对日期 / 时间范围模式的结束代码作相应调整。

EGL Java 运行时错误代码 VGJ0905E

VGJ0905E: 日期 / 时间范围模式 (声明时间戳记 / 时间间隔项的长度和日期 / 时间组件的字符串) 的起始代码或结束代码无效。

说明

日期 / 时间范围模式 (声明时间戳记 / 时间间隔项的长度和日期 / 时间组件的字符串) 的起始代码或结束代码无效。

用户响应

对日期 / 时间范围模式的起始代码或结束代码作相应调整。

EGL Java 运行时错误代码 VGJ0906E

VGJ0906E: INTERVAL 值无效。

说明

INTERVAL 值无效。

用户响应

对 INTERVAL 值作相应调整。

EGL Java 运行时错误代码 VGJ0907E

VGJ0907E: TIMESTAMP 值无效。

说明

TIMESTAMP 值无效。

用户响应

对 TIMESTAMP 值作相应调整。

EGL Java 运行时错误代码 VGJ0908E

VGJ0908E: TIME 值无效。

说明

TIME 值无效。

用户响应

对 TIME 值作相应调整。

EGL Java 运行时错误代码 VGJ0909E

VGJ0909E: DATE 值无效。

说明

DATE 值无效。

用户响应

对 DATE 值作相应调整。

EGL Java 运行时错误代码 VGJ0910E

VGJ0910E: BLOB 或 CLOB 内存不足。

说明

BLOB 或 CLOB 内存不足。

用户响应

对 BLOB 或 CLOB 大小作相应调整或将其与文件相关联。

EGL Java 运行时错误代码 VGJ0911E

VGJ0911E: 执行 loadTable 时发生了内部错误。%1

说明

执行 loadTable 时发生了内部错误。

用户响应

有关错误原因，请参阅扩展错误消息。

EGL Java 运行时错误代码 VGJ0912E

VGJ0912E: 执行 loadTable 时发生了 SQL 错误。%1

说明

执行 loadTable 时发生了 SQL 错误。

用户响应

有关错误原因，请参阅扩展错误消息。

EGL Java 运行时错误代码 VGJ0913E

VGJ0913E: 执行 loadTable 时发生了 I/O 错误。%1

说明

执行 loadTable 时发生了 I/O 错误。

用户响应

有关错误原因，请参阅扩展错误消息。

EGL Java 运行时错误代码 VGJ0914E

VGJ0914E: 装入 VGJSystemCommandProcessing 系统库时发生了错误。%1

说明

装入 VGJSystemCommandProcessing 系统库时发生了错误。

用户响应

有关错误原因，请参阅扩展错误消息。

EGL Java 运行时错误代码 VGJ0915E

VGJ0915E: 执行系统命令 %1 时发生了系统错误。检查系统的路径以了解命令是否存在以及它是否为可执行命令等等。

说明

执行系统命令时发生了错误。

用户响应

检查系统的路径以了解命令是否存在以及它是否为可执行命令等等。

EGL Java 运行时错误代码 VGJ0916E

VGJ0916E: 执行 unloadTable 时发生了内部错误。%1

说明

执行 unloadTable 时发生了内部错误。

用户响应

有关错误原因，请参阅扩展错误消息。

EGL Java 运行时错误代码 VGJ0917E

VGJ0917E: 执行 unloadTable 时发生了 SQL 错误。%1

说明

执行 unloadTable 时发生了 SQL 错误。

用户响应

有关错误原因，请参阅扩展错误消息。

EGL Java 运行时错误代码 VGJ0918E

VGJ0918E: 执行 `unloadTable` 时发生了 I/O 错误。%1

说明

执行 `unloadTable` 时发生了 I/O 错误。

用户响应

有关错误原因，请参阅扩展错误消息。

EGL Java 运行时错误代码 VGJ0920E

VGJ0920E: 从本机 C 函数返回 %1 时发生了错误。

说明

将值从 C 返回至 EGL 时发生了错误。

用户响应

这是内部错误。

EGL Java 运行时错误代码 VGJ0921E

VGJ0921E: 将 %1 传递至本机 C 函数时发生了错误。

说明

将值从 EGL 传递至 C 时发生了错误。

用户响应

这是内部错误。

EGL Java 运行时错误代码 VGJ0922E

VGJ0922E: 将本机 C 函数返回的值指定给 %1 时发生了错误。

说明

将值从 C 返回至 EGL 时发生了错误。

用户响应

这是内部错误。

EGL Java 运行时错误代码 VGJ0923E

VGJ0923E: 值太大，无法装入到 %1 中。

说明

该数字超出 `smallint` 或 `int` 接收变量的限制。

用户响应

要存储超出 `smallint` 或 `int` 范围之外的数字，重新定义该变量以使用 `int` 或十进制类型。

EGL Java 运行时错误代码 VGJ0924E

VGJ0924E: 不能在指定类型之间进行转换。

说明

在本地函数调用期间，EGL 尝试进行任何有意义的转换。但有些转换不受支持，如时间间隔与日期以及时间戳记与货币之间的转换等等。

用户响应

检查您是否指定了您期望的数据类型。

EGL Java 运行时错误代码 VGJ0925E

VGJ0925E: 自变量堆栈是空的。

说明

在将值传递至本机 C 函数或从本机 C 函数返回值时遇到了空堆栈异常。

用户响应

检查接收变量的数目是否未超出传递或返回的值的数目。

EGL Java 运行时错误代码 VGJ0926E

VGJ0926E: 内存分配失败。

说明

当前本机函数调用中的内容需要分配内存，但内存不可用。

用户响应

若干对象可能导致此错误。例如，应用程序请求的资源超出系统配置所允许的资源；或者操作系统出现问题，要求您重新引导系统。

EGL Java 运行时错误代码 VGJ0927E

VGJ0927E: 无效日期时间或时间间隔限定符。

说明

在本机 C 函数中接收时间戳记或时间间隔值时使用了无效的限定符。

用户响应

检查您是否指定了您期望的限定符。

EGL Java 运行时错误代码 VGJ0928E

VGJ0928E: 数据的字符主变量太短。

说明

在本机 C 函数中使用字符串时，使用的字符主变量不够大。

用户响应

检查变量的大小。

EGL Java 运行时错误代码 VGJ0929E

VGJ0929E: 找不到本机 C 函数 %1。

说明

在函数表中找不到指定的 C 函数。

用户响应

在函数表中添加用于此函数的条目并重新创建共享库。

EGL Java 运行时错误代码 VGJ0930E

VGJ0930E: 在本机 C 代码中对 `loc_t` 结构作了不正确的修改。

说明

已将 `Clob` 或 `Blob` 数据类型传递至本机 C 函数，但在其中接收它的 `loc_t` C 结构作了不正确的更改。

用户响应

检查 `loc_t` 结构中的 `loc_loctype`、`loc_type` 或 `loc_fname` 在本机 C 代码中是否作了更改。

EGL Java 运行时错误代码 VGJ0931E

VGJ0931E: 处理大对象时发生了错误。

说明

对 `Clob` 或 `Blob` 数据类型执行某个内部操作时发生了错误。

用户响应

这是内部错误。

EGL Java 运行时错误代码 VGJ0932E

VGJ0932E: 本机 C 函数 %1 返回的值的数目不是调用函数期望的正确数目。

说明

如果函数是作为表达式的一部分调用的，则它会返回多个值。否则返回的变量的数目与接收变量的数目不同。

用户响应

检查是否调用了正确的函数。查看本机 C 函数的逻辑（特别是它返回的值）以确保它总是返回的值的数目是期望的数目。

EGL Java 运行时错误代码 VGJ0933E

VGJ0933E: 操作的时间间隔不兼容。

说明

一些时间间隔值的组合没有意义并且是不允许的。

用户响应

查看要传递或返回的时间间隔数据类型的兼容性。

EGL Java 运行时错误代码 VGJ1000E

VGJ1000E: %1 失败。调用一个方法或者访问名为 %2 的字段导致了未处理的错误。错误消息为 %3

说明

在 Java 访问函数中发生了错误。抛出了异常，并且函数不是在 try 语句中调用的或 VGVar.handleSysLibraryErrors 为 0；或者抛出了除异常以外的其它内容，如错误。

用户响应

使用错误消息中的信息来更正问题。如果抛出了某种类型的异常，则更改程序逻辑以便通过在 try 语句内调用 Java 访问函数或通过调用 Java 访问函数前将 VGVar.handleSysLibraryErrors 设置为 1 来处理错误。

EGL Java 运行时错误代码 VGJ1001E

VGJ1001E: %1 失败。%2 不是一个标识，或者它是空对象的标识。

说明

在 Java 访问函数中发生了错误。不能使用该标识，原因是它并不表示非空对象。

用户响应

使用非空对象的标识。

EGL Java 运行时错误代码 VGJ1002E

VGJ1002E: %1 失败。名为 %2 的公用方法、字段或类不存在或不能被装入，或者参数的数目或类型不正确。错误消息为 %3

说明

找不到 Java 访问函数所使用的方法、字段或类。

用户响应

执行下列操作:

- 确保目标是公用方法、字段或类。
- 确保方法、字段或类的名称正确。必须使用类的包名来限定类名。
- 如果问题是缺少类而名称正确，则确保包含该类的目录或归档位于 Java 类路径中。
- 如果问题是缺少方法而名称正确，则应确保参数的类型和数目正确。将传递给 Java 访问函数的值与方法期望的值进行比较。

EGL Java 运行时错误代码 VGJ1003E

VGJ1003E: %1 失败。EGL 中的值的类型与 Java 中期望的 %2 的类型不匹配。错误消息为 %3

说明

传递给 Java 访问函数的值的类型不正确。

用户响应

赋值给字段的值和传递给方法和构造函数的参数必须具有正确的类型。只要类型之间的转换在 Java 中是有效的，就不需要精确匹配。例如，可使用子类而不使用其超类，并且可使用较小的基本类型（例如 short）而不使用较大的基本类型（例如，int）。

EGL Java 运行时错误代码 VGJ1004E

VGJ1004E: %1 失败。目标是返回空值的方法、不返回值的方法或空值字段。

说明

Java 访问函数期望操作结果为非空对象，但是未获得任何非空对象。

用户响应

要调用可能返回空值或不返回值的方法，使用 javaStore；或者使用 java 系统函数并且不将结果赋值给任何一个项。要获取可能为空的字段的值，使用 javaStoreField。

EGL Java 运行时错误代码 VGJ1005E

VGJ1005E: %1 失败。返回的值与返回项的类型不匹配。

说明

由于类型不匹配，所以不能将 Java 访问函数返回的值赋值给返回项。

用户响应

更改程序逻辑以使用具有适当类型的返回项。

EGL Java 运行时错误代码 VGJ1006E

VGJ1006E: %1 失败。未能装入强制类型转换为 null 的自变量的类 %2。错误消息为 %3

说明

找不到传递给 Java 访问函数的自变量的类。

用户响应

执行下列操作:

- 确保类的名称正确。必须使用包名来限定类名。
- 如果名称正确，则确保包含该类的目录或归档位于 Java 类路径中。

EGL Java 运行时错误代码 VGJ1007E

VGJ1007E: %1 失败。未能获取有关名为 %2 的方法或字段的信息，或者尝试了设置声明为 final 的字段的价值。错误消息为 %3

说明

尝试获取有关方法或字段的信息时抛出了 SecurityException 或 IllegalAccessException，或者尝试了设置声明为 final 的字段的价值。不能修改声明为 final 的字段。

用户响应

执行下列操作:

- 如果在设置值时发生了问题，则更改程序逻辑以使代码不尝试设置声明为 final 的字段的价值；或者，更改字段的声明。
- 如果访问信息时发生问题，则请系统管理员更新 Java 虚拟机的安全策略文件，以使程序具有必需的许可权。管理员可能需要授予 ReflectPermission“suppressAccessChecks”。

EGL Java 运行时错误代码 VGJ1008E

VGJ1008E: %1 失败。%2 是接口或抽象类，因此不能调用它的构造函数。

说明

不能调用接口或抽象类的构造函数。

用户响应

更改程序逻辑，以调用非抽象类的构造函数。

EGL Java 运行时错误代码 VGJ1009E

VGJ1009E: %1 失败。方法或字段 %2 不是静态的。必须使用标识而不是类名。

说明

当未将方法或字段声明为静态的时，它只存在于类的特定实例中而不是存在于类本身中。在这种情况下，必须使用对象的标识。

用户响应

更改程序逻辑以使用标识而不是类名。

EGL Java 运行时错误代码 VGJ1101W

VGJ1101W: 您要进行的方向没有其它行。

说明

最终用户尝试浏览的位置超出最后一行。

EGL Java 运行时错误代码 VGJ1148E

VGJ1148E: 操作字段“%1”不存在。

说明

当前 OnEvent 操作引用了找不到的字段。

用户响应

验证该字段在当前表单中是否存在。

EGL Java 运行时错误代码 VGJ1149E

VGJ1149E: 不能插入另一行 - 输入数组是完整的。

说明

用于容纳数组数据的变量没有空间来容纳另一行。

用户响应

增加 EGL 变量的存储空间大小。

EGL Java 运行时错误代码 VGJ1150E

VGJ1150E: 找不到数组“%1”。

说明

在 ConsoleForm 中找不到指定的数组。

用户响应

验证在 ConsoleForm 和 EGL 程序中是否正确定义了该数组。

EGL Java 运行时错误代码 VGJ1151E

VGJ1151E: 对提示结果变量赋值失败。

说明

对提示结果变量赋值失败。

用户响应

验证结果变量是否能容纳提示操作的结果。

EGL Java 运行时错误代码 VGJ1152E

VGJ1152E: 屏幕数组字段“%1”大小不正确。

说明

指定的屏幕数组字段大小不正确。

用户响应

验证屏幕数组的定义及它在 EGL 程序中的使用情况。

EGL Java 运行时错误代码 VGJ1153E

VGJ1153E: DrawBox 参数超出范围。

说明

DrawBox 参数不在当前屏幕 / 窗口维中。

用户响应

验证 drawbox 函数的参数及当前窗口维。

EGL Java 运行时错误代码 VGJ1154E

VGJ1154E: 屏幕坐标超出窗口边界。

说明

屏幕坐标超出窗口边界。

用户响应

验证要使用的坐标是否在窗口大小内。

EGL Java 运行时错误代码 VGJ1155E

VGJ1155E: 键名“%1”无效。

说明

指定键名未遵循键名约定。

用户响应

重写键名以遵循 EGL 键名约定。

EGL Java 运行时错误代码 VGJ1156E

VGJ1156E: 因为存在图片，所以不能使用此编辑功能。

说明

图片属性限制了此字段的编辑功能。

用户响应

使用备用编辑键和操作以获取期望的结果。

EGL Java 运行时错误代码 VGJ1157E

VGJ1157E: 找不到窗口“%1”。

说明

找不到该窗口。

用户响应

验证是否正确定义并使用了该窗口。

EGL Java 运行时错误代码 VGJ1158E

VGJ1158E: 新的窗口位置 / 维值无效。

说明

指定的位置 / 维值对当前显示环境无效。

用户响应

验证窗口位置 / 维值对当前显示环境是否有效。

EGL Java 运行时错误代码 VGJ1159E

VGJ1159E: 命令堆栈不同步。

说明

在 OnEvent 子句中执行的语句导致 EGL 不同步。

用户响应

验证 OnEvent 块语句中的语句 / 函数调用的使用情况。

EGL Java 运行时错误代码 VGJ1160E

VGJ1160E: 控制台用户界面库未初始化。

说明

尝试在初始化控制台用户界面库之前使用该库。

用户响应

验证控制台用户界面语句顺序是否有效。

EGL Java 运行时错误代码 VGJ1161E

VGJ1161E: 构造的字段类型是非法的。

说明

在控制台字段中指定的字段类型对构造查询操作无效。

用户响应

验证控制台字段中的字段类型对构造查询操作是否有效。

EGL Java 运行时错误代码 VGJ1162E

VGJ1162E: 不能用变量列表定义 ConstructQuery。

说明

使用变量列表调用了构造查询操作。

用户响应

验证是否正确调用了构造查询操作。

EGL Java 运行时错误代码 VGJ1163E

VGJ1163E: 只能禁用可视菜单项。

说明

尝试隐藏不可视菜单项是无效操作。

用户响应

验证要禁用的正确菜单项是否为可视菜单项。

EGL Java 运行时错误代码 VGJ1164E

VGJ1164E: 编辑操作失败。

说明

指定的编辑操作执行失败。

用户响应

验证是否正确定义了控制台字段；以及要执行的编辑操作是否为有效操作。

EGL Java 运行时错误代码 VGJ1165E

VGJ1165E: 执行热键操作时发生了错误。

说明

热键操作执行失败。

用户响应

验证指定的热键是否有效，以及语句块是否同时有效。

EGL Java 运行时错误代码 VGJ1166E

VGJ1166E: 没有要退出的活动命令。

说明

尝试退出当前命令，但该命令不存在。

用户响应

验证要在其中使用退出命令的上下文是否正确。

EGL Java 运行时错误代码 VGJ1167E

VGJ1167E: 没有要继续的活动命令。

说明

尝试继续执行当前命令。

用户响应

验证要在其中使用 `continue` 语句的上下文是否正确。

EGL Java 运行时错误代码 VGJ1168E

VGJ1168E: 致命错误: %1

说明

发生了致命运行时错误。

用户响应

验证在其中使用控制台用户界面语句的上下文是否正确以及使用顺序是否正确。

EGL Java 运行时错误代码 VGJ1169E

VGJ1169E: 字段“%1”不存在。

说明

指定的控制台字段不存在。

用户响应

验证是否在控制台表单中正确定义了控制台字段。

EGL Java 运行时错误代码 VGJ1170E

VGJ1170E: 屏幕数组字段“%1”不是数组。

说明

控制台表单中的引用控制台字段不是数组。

用户响应

验证控制台字段是否定义为数组；验证要定义的控制台字段是否正确。

EGL Java 运行时错误代码 VGJ1171E

VGJ1171E: 找不到字段“%1”。

说明

找不到指定的控制台字段。

用户响应

验证是否在控制台表单中正确定义了控制台字段。

EGL Java 运行时错误代码 VGJ1172E

VGJ1172E: 不能在窗口之外创建 **ConsoleField**。

说明

尝试在控制台表单 / 窗口上下文之外创建控制台字段。

用户响应

验证控制台表单和控制台字段定义是否正确。

EGL Java 运行时错误代码 VGJ1173E

VGJ1173E: 数组字段计数不匹配。

说明

指定的控制台用户界面数组字段与引用的 EGL 数组不匹配。

用户响应

验证 **ConsoleField** 和数组定义；验证要在 `openui` 语句上使用的 EGL 数组变量是否正确。

EGL Java 运行时错误代码 VGJ1174E

VGJ1174E: 表单“%1”不存在。

说明

指定的控制台表单不存在。

用户响应

验证在其中定义和使用指定的控制台表单的上下文是否正确。

EGL Java 运行时错误代码 VGJ1175E

VGJ1175E: 表单“%1”不能装入到窗口“%2”中。

说明

该表单的维导致该表单无法装入到当前窗口维中。

用户响应

改变表单定义或窗口定义的维。

EGL Java 运行时错误代码 VGJ1176E

VGJ1176E: 字段列表不匹配。

说明

指定字段列表包含的项数与提供的变量列表中的项数不同。

用户响应

改变 `openUI` 语句以确保指定的字段数和变量数相同。

EGL Java 运行时错误代码 VGJ1177E

VGJ1177E: 表单“%1”正忙。

说明

目前已在另一上下文中使用表单引用。

用户响应

验证 EGL 程序逻辑是否一次只使用一个表单。

EGL Java 运行时错误代码 VGJ1178E

VGJ1178E: 已使用表单名“%1”。

说明

表单定义导致表单名冲突。

用户响应

改变表单定义以使用唯一表单名。

EGL Java 运行时错误代码 VGJ1179E

VGJ1179E: 表单“%1”未打开。

说明

尝试引用未定义的表单对象。

用户响应

验证是否正确定义了指定表单，并且使用它的 `ConsoleUI` 语句是否有效。

EGL Java 运行时错误代码 VGJ1180E

VGJ1180E: 不能在窗口不存在的情况下创建 `ConsoleForm`。

说明

尝试在没有有效窗口引用的情况下创建 `ConsoleForm`。

用户响应

验证是否正确定义了 ConsoleForm，并且使用它的 ConsoleUI 语句是否有效。

EGL Java 运行时错误代码 VGJ1181E

VGJ1181E: 不能将 `KeyObject.getChar()` 用于虚拟键。

说明

consoleUI 不能将 `KeyObject.getChar()` 用于虚拟键。

用户响应

改变 EGL 程序以构造用于虚拟键定义的字符串。

EGL Java 运行时错误代码 VGJ1182E

VGJ1182E: 不能将 `KeyObject.getCookedChar()` 用于虚拟键。

说明

ConsoleUI 不能将 `KeyObject.getCookedChar()` 用于虚拟键。

用户响应

改变 EGL 程序以使用字符串来定义虚拟键。

EGL Java 运行时错误代码 VGJ1183E

VGJ1183E: 检索提示结果字符串失败。

说明

检索提示结果字符串失败。

用户响应

EGL Java 运行时错误代码 VGJ1184E

VGJ1184E: 在资源束“%2”中找不到帮助消息键“%1”。

说明

在指定的消息帮助文件中找不到帮助消息键。

用户响应

验证要使用的帮助消息键和帮助消息文件是否正确。

EGL Java 运行时错误代码 VGJ1185E

VGJ1185E: 非法数组下标。

说明

尝试引用无效的数组元素。

用户响应

验证程序逻辑引用的数组元素是否在已定义数组的大小内。

EGL Java 运行时错误代码 VGJ1186E

VGJ1186E: 不能初始化控制台用户界面库。 .

说明

在程序启动时未能初始化控制台用户界面库。

用户响应

验证是否在受支持的显示环境和平台中使用该程序。

EGL Java 运行时错误代码 VGJ1187E

VGJ1187E: INTERNAL ERROR

说明

遇到了 ConsoleUI INTERNAL ERROR。

用户响应

EGL Java 运行时错误代码 VGJ1188E

VGJ1188E: 接收到 INTERRUPT 信号。

说明

接收到 INTERRUPT 信号。

用户响应

EGL Java 运行时错误代码 VGJ1189E

VGJ1189E: 不可视菜单项必须有加速键。

说明

尝试创建没有加速键的不可视菜单项。

用户响应

改变菜单项定义以便为不可视菜单项定义加速键。

EGL Java 运行时错误代码 VGJ1190E

VGJ1190E: 不能在窗口不存在的情况下创建 `ConsoleLabel`。

说明

在创建 `ConsoleLabel` 时找不到有效的窗口引用。

用户响应

验证是否在控制台表单中正确定义了控制台标注并且在 EGL 程序中正确使用了该控制台表单。

EGL Java 运行时错误代码 VGJ1191E

VGJ1191E: 菜单项 %1 不能装入到窗口中。

说明

指定的菜单项太大，无法装入到当前活动窗口中。

用户响应

改变菜单项以使名称小于当前活动窗口宽度。

EGL Java 运行时错误代码 VGJ1192E

VGJ1192E: 菜单项“%1”不存在。

说明

指定的菜单项找不到或不存在。

用户响应

验证是否定义了引用的菜单项并且已将其添加至当前菜单实例。

EGL Java 运行时错误代码 VGJ1193E

VGJ1193E: 菜单助记符冲突 (键 = %1)。

说明

当前菜单项定义导致助记符冲突。

用户响应

改变菜单项以确保该加速键 / OnEvent 键没有冲突。

EGL Java 运行时错误代码 VGJ1194E

VGJ1194E: 没有活动表单。

说明

控制台用户界面没有活动表单引用。

用户响应

验证是否定义并显示了表单。

EGL Java 运行时错误代码 VGJ1195E

VGJ1195E: 必须具有用于 **DISPLAY ARRAY** 的活动表单。

说明

尝试从不存在的当前活动表单中显示数组。

用户响应

尝试显示数组之前验证是否已经使用数组定义了表单。

EGL Java 运行时错误代码 VGJ1196E

VGJ1196E: 必须具有用于 **READ ARRAY** 的活动表单。

说明

尝试从不存在的当前活动表单中读取数组。

用户响应

尝试从中读取数组之前验证是否已经定义了表单并且使其处于活动状态。

EGL Java 运行时错误代码 VGJ1197E

VGJ1197E: 不能在当前命令的情况下启动事件循环。

说明

不能在当前命令的情况下启动事件循环。

用户响应

EGL Java 运行时错误代码 VGJ1198E

VGJ1198E: 未指定 **blob** 编辑器。

说明

尝试编辑 **blob**，但未指定任何 **blob** 编辑器。

用户响应

在 **blob** 控制台字段中定义相应的编辑器。

EGL Java 运行时错误代码 VGJ1199E

VGJ1199E: INTERNAL ERROR: 没有格式对象

说明

INTERNAL ERROR: 没有格式对象

用户响应

EGL Java 运行时错误代码 VGJ1200E

VGJ1200E: 没有指定帮助文件。

说明

已接收帮助请求，但未指定帮助文件。

用户响应

在 EGL 程序中定义有效帮助文件。

EGL Java 运行时错误代码 VGJ1201E

VGJ1201E: 未指定帮助消息。

说明

已接收帮助请求，但未指定帮助消息。

用户响应

改变 EGL 程序以提供帮助消息。

EGL Java 运行时错误代码 VGJ1202E

VGJ1202E: 无法使用菜单。

说明

尝试使用未显示的菜单功能。

用户响应

在显示菜单后验证菜单功能是否可用。

EGL Java 运行时错误代码 VGJ1203E

VGJ1203E: 没有当前屏幕数组。

说明

创建引用以使用不存在的当前屏幕数组。

用户响应

验证当前活动表单是否包含屏幕数组。

EGL Java 运行时错误代码 VGJ1204E

VGJ1204E: 没有可视的菜单项。

说明

在构造菜单时，找不到可视的菜单项。

用户响应

改变菜单创建过程以使至少有一个菜单项可视。

EGL Java 运行时错误代码 VGJ1205E

VGJ1205E: 新窗口的名称为空。

说明

窗口的声明为空。

用户响应

在声明窗口时提供窗口名。

EGL Java 运行时错误代码 VGJ1206E

VGJ1206E: 尝试打开空窗口。

说明

尝试打开不存在的窗口或空窗口。

用户响应

验证打开窗口语句是否在使用有效的窗口引用。

EGL Java 运行时错误代码 VGJ1207E

VGJ1207E: 提示中发生了异常。

说明

执行提示时发生了异常。

用户响应

验证提示 OnEvent 语句块是否正确。

EGL Java 运行时错误代码 VGJ1208E

VGJ1208E: 接收到 QUIT 信号。

说明

接收到 QUIT 信号。

用户响应

EGL Java 运行时错误代码 VGJ1209E

VGJ1209E: 没有活动屏幕数组。

说明

当前活动表单不包含屏幕数组。

用户响应

验证程序逻辑是否在使用包含屏幕数组定义的表单。

EGL Java 运行时错误代码 VGJ1210E

VGJ1210E: 没有活动表单。

说明

当前控制台用户界面会话不包含活动表单实例。

用户响应

在引用表单之前验证是否定义并显示了表单。

EGL Java 运行时错误代码 VGJ1211E

VGJ1211E: 菜单不能滚动至当前项。

说明

尝试将菜单光标移至菜单项失败。

用户响应

验证菜单逻辑是否正确地将菜单光标移至正确的菜单项。验证是否未显示菜单项。

EGL Java 运行时错误代码 VGJ1212E

VGJ1212E: 未知属性“%1”

说明

未识别指定属性。

用户响应

验证用于当前控制台用户界面上下文的属性是否正确。

EGL Java 运行时错误代码 VGJ1213E

VGJ1213E: 字段“%1”出现错误。

说明

字段中的输入内容不正确。

用户响应

验证输入的数据与字段的数据类型或格式属性是否匹配。

EGL Java 运行时错误代码 VGJ1214E

VGJ1214E: 未提供足够的变量。

说明

未向 openUI 语句提供足够的变量，无法与控制台表单绑定。

用户响应

改变 EGL 程序以列示用于 openUI 语句的更多变量；改变 openUI 语句以限制控制台字段的数目。

EGL Java 运行时错误代码 VGJ1215E

VGJ1215E: 已经在使用窗口名“%1”。

说明

另一窗口已在使用新定义的窗口名。

用户响应

改变窗口的名称以便不与其它窗口名发生冲突。

EGL Java 运行时错误代码 VGJ1216E

VGJ1216E: 窗口大小太小，无法显示帮助屏幕。

说明

尝试在显示环境中显示帮助屏幕，但显示环境太小，无法显示帮助屏幕。

用户响应

调整显示环境的大小。

EGL Java 运行时错误代码 VGJ1217W

VGJ1217W: 您要进行的方向没有其它字段。

说明

尝试将光标移过字段列表的结尾。

用户响应

EGL Java 运行时错误代码 VGJ1218E

VGJ1218E: 屏幕数组“%1”内容无效。

说明

对于屏幕上的每一列，屏幕上的 `arrayDictionary` 都包含一个条目。所有条目必须为同一类型的对象：或者是一个 `ConsoleField`，或者是 `ConsoleField` 数组，并且所有数组（如果有的话）必须具有相同数目的元素。

检查并更正屏幕上的 `arrayDictionary` 的声明。

EGL Java 运行时错误代码 VGJ1219E

VGJ1219E: 屏幕数组“%1”不能包含分段字段“%2”。

说明

不应在屏幕上的 `arrayDictionary` 中使用的任何 `consoleField` 中设置 `consoleField segment` 属性。

用户响应

在屏幕上的 `arrayDictionary` 中包括的任何 `consoleField` 中除去 `segment` 属性。

EGL Java 运行时错误代码 VGJ1220E

VGJ1220E: 屏幕数组“%1”与数据数组不兼容。

说明

对于屏幕上的每一列，屏幕上的 `arrayDictionary` 都包含一个条目，并且动态数组与该 `arrayDictionary` 绑定。`arrayDictionary` 中的每个记录具有的字段数目必须与屏幕上的 `arrayDictionary` 中的列数相同，并且每个字段在赋值时必须与相应的 `consoleField` 兼容。

用户响应

检查并更正屏幕上的 `arrayDictionary` 或与该 `arrayDictionary` 绑定的动态数组中的记录。

EGL Java 运行时错误代码 VGJ1221E

VGJ1221E: 检测到具有相同名称“%1”的字段。

说明

`consoleForm` 不能包含多个同名字段。

用户响应

更正 `consoleForm` 声明。

EGL Java 运行时错误代码 VGJ1222E

VGJ1222E: 控制台字段“%1”长度无效。

说明

`consoleField` 的长度必须大于零。

用户响应

更正 `consoleField` 声明。

EGL Java 运行时错误代码 VGJ1223E

VGJ1223E: [%1, %2] 处的标签未能装入到提供的空间中。

说明

标签必须能够完整地装入到窗口的边界内。

用户响应

更正标签或窗口的位置或大小。

EGL Java 运行时错误代码 VGJ1224E

VGJ1224E: (%2, %3) 处的字段“%1”未能装入到提供的空间中。

说明

`consoleField` 必须能够完整地装入到窗口的边界内。

用户响应

更正 `consoleField` 或窗口的位置或大小。

EGL Java 运行时错误代码 VGJ1225E

VGJ1225E: 提示字符串太长，无法全部显示在活动窗口中。

说明

提示用户输入的消息必须能够装入到活动窗口中。如果提示的 `isChar` 属性设置为 `no`，您还必须考虑用户响应所需的空間。

用户响应

更正提示或窗口的声明。

EGL Java 运行时错误代码 VGJ1226E

VGJ1226E: OpenUI 数组自变量无效。

说明

对于屏幕上的每一列，屏幕上的 `arrayDictionary` 都包含一个条目。所有条目必须为同一类型的对象：或者是一个 `ConsoleField`，或者是 `ConsoleField` 数组，并且所有数组（如果有的话）必须具有相同数目的元素。

用户响应

检查并更正屏幕上的 `arrayDictionary` 的声明。

EGL Java 运行时错误代码 VGJ1227E

VGJ1227E: OpenUI 字段自变量无效。

说明

在 `openUI` 语句中，可通过指定 `consoleField` 或指定包含 `consoleField` 名称字段的值的字符串来指定 `consoleField` 列表。在此情况下，该列表包括无效值。

用户响应

检查并更正 `openUI` 语句中的值。

EGL Java 运行时错误代码 VGJ1228E

VGJ1228E: 只有单个变量可以与提示语句绑定。

说明

提示只能与接收用户响应的单个变量绑定。您尝试进行其它类型的绑定。

用户响应

检查并更正 `openUI` 语句。

EGL Java 运行时错误代码 VGJ1229E

VGJ1229E: 未能确定控制台字段“%1”的数据绑定。

说明

如果 `openUI` 语句未将 `consoleField` 与其它变量绑定，则将使用每个 `consoleField` 中的 `binding` 属性的值；但在此情况下未提供 `binding` 属性。

用户响应

在 consoleField 或 openUI 语句中添加绑定。

EGL Java 运行时错误代码 VGJ1290E

VGJ1290E: %1 对于 Blob/Clob 函数不是有效的参数。

说明

处理 Blob/Clob 函数时发生了错误。错误的原因将在消息插入内容中描述。

用户响应

根据消息插入内容采取相应的操作。

EGL Java 运行时错误代码 VGJ1301E

VGJ1301E: 报告填写错误 %1。

说明

报告填写错误。提供给报告的数据不正确。原因可能是动态数组记录字段名与报告字段名不匹配、连接不存在或者 SQL 语句无效。

用户响应

如果要使用记录的动态数组，请确保报告设计中定义的字段名与记录中的元素相匹配（按名称）。如果要使用 SQL 语句，确保 SQL 有效。如果要使用连接，确保已建立连接并且连接名称是正确的。此外，确保为 reportDesignFile 指定的路径名有效并且该文件存在。

EGL Java 运行时错误代码 VGJ1302E

VGJ1302E: 报告导出错误 %1。

说明

报告导出错误。报告未能导出至指定格式。

用户响应

确保路径名是正确的，填写的报告对象在指定位置存在并且在 reportDestinationFile 字段中进行了正确的赋值。

EGL Java 运行时错误代码 VGJ1303E

VGJ1303E: 报告动态访问错误，找不到内容。%1

说明

该字段名在记录的动态数组中不存在。

用户响应

确保字段名与报告设计和您要在 EGL 程序中使用的记录中是匹配的。

EGL Java 运行时错误代码 VGJ1304E

VGJ1304E: 不正确的连接名称

说明

连接名称无效。

用户响应

确保连接是有效的 EGL 连接并且 `defineDatabaseAlias` 函数已用于对连接指定名称。

EGL Java 运行时错误代码 VGJ1305E

VGJ1305E: 带有指定名称 %1 的连接不存在

说明

带有该连接名称的连接不存在。

用户响应

确保下列语句在 EGL 程序中存在。带有有效参数的连接函数和对连接指定名称的 `defineDatabaseAlias`。

EGL Java 运行时错误代码 VGJ1306E

VGJ1306E: 不正确的 EGL 和报告类型映射。检查映射表。

说明

报告设计中的字段与 EGL 程序中的数据类型之间存在类型不匹配的问题。

用户响应

确保这些类型是兼容的，就像文档中提到的那样。一些示例：对于 EGL `char` 类型，设计文件应该将该类定义为 `java.lang.String`，对于 EGL `int` 类型，设计文件应该将字段类定义为 `java.lang.Integer`。

EGL Java 运行时错误代码 VGJ1401E

VGJ1401E: 位置 (%2,%3) 处的字段“%1”不在表单内。

说明

给定位置的指定字段不在表单内。

用户响应

验证是否正确定义了表单和字段。

EGL Java 运行时错误代码 VGJ1402E

VGJ1402E: 字段“%1”与“%2”重叠。

说明

两个字段的大小和位置导致它们重叠在一起。

用户响应

调整字段的大小和位置坐标。

EGL Java 运行时错误代码 VGJ1403E

VGJ1403E: 内部错误: 不能确定表单组。

说明

内部错误: 不能确定表单组。

用户响应

验证是否正确定义了表单和表单组。

EGL Java 运行时错误代码 VGJ1404E

VGJ1404E: 表单“%1”不能装入到任何浮动区域中。

说明

该表单不能装入到任何浮动区域中。

用户响应

验证该表单是否能正确显示在浮动区域中。

EGL Java 运行时错误代码 VGJ1405E

VGJ1405E: 字段“%1”坐标无效。

说明

字段坐标无效。

用户响应

验证指定字段坐标对该表单是否有效。

EGL Java 运行时错误代码 VGJ1406E

VGJ1406E: 不能获取打印关联。

说明

尝试设置打印关联失败。

用户响应

验证打印机关联的设置是否正确。

EGL Java 运行时错误代码 VGJ1407E

VGJ1407E: 没有合适的打印设备大小。

说明

没有合适的打印设备大小。

用户响应

EGL Java 运行时错误代码 VGJ1408E

VGJ1408E: 找不到打印机“%1”。下列打印机可用: %2

说明

用户尝试打印至特定打印机设备，但在系统中找不到该设备。

用户响应

检查环境中的打印机配置。确保该打印机存在，或者打印至另一打印机。

EGL Java 运行时错误代码 VGJ1409E

VGJ1409E: 没有用于表单的显示设备。

说明

没有用于表单的显示设备。

用户响应

验证 EGL 程序是否在受支持的平台和显示环境中执行。

EGL Java 运行时错误代码 VGJ1410E

VGJ1410E: 没有用于已显示表单的兼容设备大小。

说明

没有用于已显示表单的兼容设备大小。

用户响应

验证 EGL 程序是否在受支持的平台和显示环境中执行。

EGL Java 运行时错误代码 VGJ1411E

VGJ1411E: 帮助表单类“%1”不存在。

说明

尝试引用不存在的帮助表单类。

用户响应

验证是否正确定义并引用了帮助表单类。

EGL Java 运行时错误代码 VGJ1412E

VGJ1412E: 未知属性“%1”。

说明

未识别指定属性。

用户响应

验证是否在使用正确的属性名称。

EGL Java 运行时错误代码 VGJ1414E

VGJ1414E: 不能创建帮助表单“%1”

说明

未能创建要显示的帮助表单。

用户响应

确保相应的表单组和表单在应用程序中存在并且是以正确的方式生成的。

EGL Java 运行时错误代码 VGJ1415E

VGJ1415E: 内部错误: %1

说明

发生了内部错误。

用户响应

请与 IBM 技术支持机构联系。

EGL Java 运行时错误代码 VGJ1416E

VGJ1416E: 没有可用的打印机。

说明

用户尝试进行打印，但系统没有可用的打印机设备。

用户响应

改为打印至文件，或者在环境中配置打印机设备。

EGL Java 运行时错误代码 VGJ1417E

VGJ1417E: 没有缺省打印机。

说明

用户尝试打印至缺省打印机，但系统中未指定缺省打印机。

用户响应

打印至特定打印机，或在环境中定义缺省打印机。

EGL Java 运行时错误代码 VGJ1419E

VGJ1419E: “%2”的定义格式的值“%1”比允许的最大长度（%3）要长。

说明

定义格式的值（日期、时间和货币）太长，无法装入到字段中。

用户响应

增加字段的大小，或者将该值的格式定义为较短长度。

EGL Java 运行时错误代码 VGJ9900E

VGJ9900E: 发生了错误。错误为 %1。无法装入错误描述。

说明

程序找不到或无法装入缺省消息类文件以及语言环境的消息类文件。这两个消息类文件的其中一个或两个可能已丢失或损坏。

注: 在运行时，可能会由于装入消息文件时发生问题而只能使用美国英语来显示此消息。

用户响应

如果已经从文件 fda6.jar 中解压缩类文件，则验证您所拥有的类是否与最近的文件中的类处于同一发行版或维护级别。如果您正在使用较旧版本的类，则将它们替换为正确的版本。并且，可以从 EGL 重新安装 fda6.jar。

如果问题仍然存在，则执行以下操作：

1. 记录消息号和消息文本。

注： 错误消息包括下列重要信息：

- 错误的发生位置
- 内部错误的类型

2. 记录发生此消息的情况。

3. 有关如何向“IBM 支持中心”报告可能存在的缺陷的更多指示信息，参阅 *EGL Installation Guide*。

EGL Java 运行时错误代码 VGJ9901E

VGJ9901E: 发生了错误。错误为 %1。在消息类文件 %2 中找不到 %1 的消息文本。也找不到 VGJ0002E 的消息文本。

说明

消息类文件未包含该消息标识或消息标识 VGJ0002E 的运行时消息。消息类文件已损坏，或是来自 EGL 的前发行版。

注： 在运行时，可能会由于装入消息文件时发生问题而只能使用美国英语来显示此消息。

用户响应

如果已经从文件 `fda.jar` 解压缩类文件，则验证您拥有的类是否与该 JAR 文件中的类处于同一发行版或维护级别。如果您正在使用较旧版本的类，则将它们替换为正确的版本。并且，可以从 EGL 重新安装 `fda6.jar`。

如果问题仍然存在，则执行以下操作：

1. 记录消息号和消息文本。

注： 错误消息包括下列重要信息：

- 错误的发生位置
- 内部错误的类型

2. 记录发生此消息的情况。

3. 有关如何向“IBM 支持中心”报告可能存在的缺陷的更多指示信息，参阅 *EGL Installation Guide*。

附录. 声明

Note to U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

本信息是为在美国提供的产品和服务编写的。IBM 可能在其他国家或地区不提供本文中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代表咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可。您可以用书面方式将许可查询寄往：

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

有关双字节（DBCS）信息的许可查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往：

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区： International Business Machines Corporation “按现状”提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销和适用于某种特定用途的保证。某些国家或地区在某些交易中不允许免除明示或暗含的保证。因此本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本资料的新版本中。IBM 可以随时对本资料中描述的产品和 / 或程序进行改进和 / 或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。那些 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的：（i）允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及（ii）允许对已经交换的信息进行相互使用，请与下列地址联系：

IBM Corporation
Lab Director
IBM Canada Ltd. Laboratory
8200 Warden Avenue
Markham, Ontario, Canada L6G 1C7

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际软件许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其他操作环境中获得的数据可能会有明显的不同。有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量是通过推算而估计的，实际结果可能会有差异。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

所有 IBM 的价格均是 IBM 当前的建议零售价，可随时更改而不另行通知。经销商的价格可与此不同。

本信息仅用于规划用途。在提供所描述的产品之前，此处的信息可以更改。

本信息包含日常商业运作所使用的数据和报表的示例。为了尽可能全面地说明这些数据和报表，这些示例包括个人、公司、商标和产品的名称。所有这些名称都是虚构的，如与实际商业企业所使用的名称和地址有任何雷同，纯属巧合。

版权许可：

本信息包括源语言形式的样本应用程序，这些样本说明不同操作平台上的编程方法。如果是为按照在编写样本程序的操作平台上的应用程序编程接口（API）进行应用程序的开发、使用、经销或分发为目的，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。这些示例并未在所有条件下作全面测试。因此，IBM 不能担保或暗示这些程序的可靠性、可维护性或功能。用户如果是为了按照 IBM 应用程序编程接口开发、使用、经销或分发应用程序，则可以任何形式复制、修改和分发这些样本程序，而无须向 IBM 付费。

凡这些实例程序的每份拷贝或其任何部分或任何衍生产品，都必须包括如下版权声明：

©（贵公司的名称）（年）。此部分代码是根据 IBM 公司的样本程序衍生出来的。
© Copyright IBM Corp. 2000, 2004. All rights reserved.

如果您正以软拷贝格式查看本信息，图片和彩色图例可能无法显示。

编程接口信息

编程接口信息旨在帮助您创建使用此程序的应用软件。

通用编程接口允许您编写获取此程序的工具服务的应用软件。

但是，此信息也可能包含诊断、修改和调整信息。这些诊断、修改和调整信息用于帮助您调试应用软件。

警告： 不要将此诊断、修改和调整信息用作编程接口，因为它可能会更改。

商标和服务标记

下列各项是 International Business Machines Corporation 在美国和 / 或其它国家或地区的商标：

- AIX
- CICS
- CICS/ESA®
- ClearCase
- DB2
- IBM
- IMS
- Informix
- iSeries
- MQSeries
- MVS™
- OS/400
- RACF®
- Rational
- VisualAge
- WebSphere
- z/OS

Intel 是 Intel Corporation 在美国和 / 或其他国家或地区的商标。

Java 和所有基于 Java 的商标是 Sun Microsystems, Inc. 在美国和 / 或其他国家或地区的商标。

Linux 是 Linus Torvalds 在美国和 / 或其他国家或地区的商标。

Microsoft®、Windows 和 Windows NT 是 Microsoft Corporation 在美国和 / 或其他国家或地区的商标。

UNIX 是 The Open Group 在美国和其他国家或地区的注册商标。

其他公司、产品或服务名称可能是其他公司的商标或服务标记。

索引

[A]

安装, 用于 Java 的 EGL 运行时编码
318

[B]

绑定 175

包

创建 117
建议 13
描述 13

保留字

EGL 444

报告

报告处理程序 193
报告处理程序的代码示例 201
编写报告驱动程序代码 205
创建 190
创建和生成概述 190
导出 207
概述 189
库 788
模板 199
驱动程序函数的代码示例 197
数据源 191
数据源样本代码 197
添加设计文档 198
已导出文件格式 207
用于调用报告的代码 205
在创建后生成 206
XML 设计文档 189
XML 设计文档中的数据类型 196

报告处理程序

创建 200, 201
代码示例 201
概述 193
函数 194
可以调用的函数 195

报告的设计文档

概述 189
其中的数据类型 196
添加至包 198

报告的数据源 191

报告库

报告记录 192
概述 788
ReportData 记录 192

编辑器

打开部件 252
定位源文件 253

编辑器 (续)

内容辅助 118, 441
首选项, EGL 108
EGL 441
变量, 声明 49
变量, 引用 53
标准 JDBC 连接 240
表达式
逻辑 80, 454
描述 452
日期时间 453
数字 80, 461
文本 462
string 80
表单部件
编辑 151
创建打印表单 143
创建文本表单 145
定义格式的属性 61
过滤 152, 161
描述 142
模板 156
验证属性 62
在 EGL 表单编辑器中创建表单 152
字段显示属性 60
EGL 源格式 467
print 144
text 145
别名
概述 608
Java 610
Java 包装器 610
部件
打开 252
描述 16
属性 59
搜索 251
引用 20
部件属性
ConsoleForm 413
部署描述符
更新 324
设置值 322
部署设置, J2EE
描述符值 322, 324
运行时环境 321
ConnectionFactory, CICSJ2C 325
JDBC 连接 328
TCP/IP 侦听器 320, 326
部署, J2EE 外部的 Java 应用程序 318

[C]

参数, 程序 663

参数, 函数 477

常量, 声明 49

常量, 引用 53

程序部件

参数 663

创建 127

非参数数据 660

基本 666

描述 128

使用声明 875

输入表单 672

输入记录 672

属性 670

EGL 源格式 664

Java 包装器生成 615

Java 程序生成 297

Java 生成 614

textUI 668

程序调用 8

程序属性文件 317

程序转移 8

重建项目菜单选项 295

初始化, 数据 430

处理程序部件

创建 200

串行记录部件 679

创建 164

从 EGL 接收值 395

[D]

打印表单 144

代码段

插入 137

autoRedirect 138

databaseUpdate 139

getClickedRowValue 139

setCursorFocus 138

代码生成, 类型 8

带索引记录部件 488

调试器, EGL

查看变量 266

创建启动配置 261

创建侦听器启动配置 262

从生成的代码中调用 255

单步执行程序 265

概述 255

构建描述符 255

- 调试器, EGL (续)
 - 建议 255
 - 命令 255
 - 启动程序 260
 - 启动服务器 263
 - 启动 Web 会话 263
 - 设置首选项 106
 - 使用断点 264
 - 系统类型首选项 255
 - 准备服务器 262
 - call 语句 255
 - SQL 数据库访问 255

调用

- C 函数 390

- 定义格式的属性 61

- 动态数组 68

- 动态 SQL 语句 219

- 断点 264

- 多维数组 68

[F]

分段

- 文本应用程序 147

- 赋值 340

- 赋值兼容性 335

[G]

- 功能, 启用 113

- 工作台, 生成于 298, 300

构建部件

- 构建描述符 108, 267

- 链接选项 282

- 资源关联 277

构建服务器

- 描述 311

- 在 AIX、Linux 或 Windows

- 2000/NT/XP 上启动 311

构建规划

- 描述 297

- 在生成之后调用 303

构建脚本

- 必需选项 369

- 描述 310

构建路径, EGL

- 编辑 290

- 概述 435

构建描述符部件

- 编辑一般选项 272

- 编辑 Java 运行时属性 275

- 除去 276

- 描述 267

- 设置缺省值 108

- 添加 271

构建描述符部件 (续)

- 选项, 字母列表 347

- 主构建描述符 270

- Java 选项 273

构建文件

- 编辑 import 语句 289

- 创建 267

- 描述 13

- 添加 import 语句 289, 290

- format 346

构建项目菜单选项 295

固定记录部件

- 描述 123

关键字

- new 168

关键字语句

- 按字母顺序列示的列表 83

- 与 MQSeries 相关的 244

- 字母列表 83

- add 511

- call 513

- case 515

- close 517

- continue 519

- converse 519

- delete 520

- display 522

- execute 522

- exit 526

- for 528

- forEach 530

- forward 531

- freeSQL 532

- get 533

- get absolute 538

- get current 540

- get first 541

- get last 543

- get next 544

- get previous 549

- get relative 552

- goTo 554

- if, else 555

- move 556

- open 561

- prepare 574

- print 576

- replace 576

- return 579

- set 579

- show 588

- transfer 589

- try 590

- while 591

关联元素 340

[H]

- 函数部件 130, 481

- 变量 475

- 参数 477

- 创建 129

- 函数调用 473

- 函数, Java 访问 737

- 环境文件, J2EE

- 更新 323

- 描述 324

[J]

- 基本程序部件 666

- 基本记录部件 345

基本类型

- 描述 31

- ANY 34

- BIN 46

- BLOB 45

- CHAR 35

- CLOB 44

- DATE 38

- DBCHAR 35

- DECIMAL 46

- FLOAT 47

- HEX 36

- INTERVAL 38

- MBCHAR 36

- MONEY 47

- NUM 47

- NUMC 48

- PACF 48

- Page Designer 181

- SMALLFLOAT 49

- STRING 37

- TIME 40

- TIMESTAMP 40

- UNICODE 37

基本应用程序

- 启动 308

基本字段级别属性 625

- action 628

- align 629

- byPassValidation 629

- color 630

- column 631

- currency 632

- currencySymbol 632

- dateFormat 633

- displayName 635

- displayUse 635

- fieldLen 636

- fill 637

- fillCharacter 637

基本字段级别属性 (续)

- help 637
- highlight 638
- inputRequired 638
- inputRequiredMsgKey 638
- intensity 639
- isBoolean 639
- isDecimalDigit 639
- isHexDigit 640
- isNullable 640
- isReadOnly 641
- lineWrap 642
- lowerCase 642
- masked 642
- maxLength 643
- minimumInput 643
- minimumInputMsgKey 644
- modified 644
- needsSOSI 644
- newWindow 645
- numElementsItem 646
- numericSeparator 646
- outline 646
- pattern 647
- persistent 647
- protect 648
- selectFromListItem 649
- selectType 649
- sign 650
- sqlDataCode 650
- sqlVariableLen 651
- timeFormat 652
- timeStampFormat 653
- typeChkMsgKey 654
- upperCase 654
- validationOrder 655
- validatorDataTable 655
- validatorDataTableMsgKey 656
- validatorFunction 656
- validatorFunctionMsgKey 657
- validValues 658
- validValuesMsgKey 659
- value 659
- zeroFormat 660

记录部件

- 串行 679
- 创建 122
- 带索引 488
- 基本 345
- 描述 122
- 属性, 可变量长度 673
- 相关 676
- MQ 603
- Page Designer 182
- SQL 209, 232, 683

记录类型

- 描述 125
- 与文件类型相关联 673
- 记录内部结构, SQL 682
- 检索功能, SQL 209, 231
- 检索首选项, SQL 112
- 键盘快捷键 119
- 建议, 开发
 - 包 13
 - 部件指定 13
 - 构建描述符 13
- 将值返回至 EGL 397
- 结构 23
- 结构字段数组 71
- 结果集处理, SQL 209, 678
- 结果视图, 生成 485
- 结果文件 298
- 静态数组 68

[K]

- 开发过程 7
- 控制台用户界面变量
 - errorLine 707
- 控制台用户界面概述 163
- 库部件
 - 创建 130
 - 生成的输出 591
 - 使用说明 875
 - EGL 源格式 592
- 库部件, 类型 basicLibrary
 - 描述 131
- 库部件, 类型 nativeLibrary
 - 描述 132

[L]

- 类型定义 25
- 链接属性文件
 - 部署 329
 - 描述 330
 - 详细信息 598
- 链接选项部件
 - 编辑与转移相关的元素 287
 - 编辑 asynchLink 元素 286
 - 编辑 callLink 元素 285
 - 除去 288
 - 描述 282
 - 添加 284
- 逻辑表达式 454
- 逻辑部件
 - 函数 130, 481
 - 基本程序 666
 - 库 592
 - 库, 类型 basicLibrary 131

逻辑部件 (续)

- 库, 类型 nativeLibrary 132
- PageHandler 177
- pageHandler 619
- textUI 程序 668
- 逻辑工作单元 279

[M]

- 名称
 - 别名 608, 610
 - 约定 612
- 命令文件 439
- 模板, 首选项 109
- 目录, 生成到 304

[N]

- 内容辅助
 - 描述 441
 - 使用 118

[P]

- 片段
 - 插入 137
 - autoRedirect 138
 - databaseUpdate 139
 - getClickedRowValue 139
 - setCursorFocus 138

[Q]

- 启动配置
 - 显式 261
 - 隐式 260
 - 侦听器 262
- 全部重建菜单选项 295
- 缺省数据库, SQL 230

[R]

- 日期时间表达式 453
- 日期、时间和时间戳记格式说明符 42

[S]

- 商标 991
- 生成
 - 概述 293
 - 工作台 300
 - 结果视图 485
 - 库部件 591
 - 目录目标 304

生成 (续)

- 批处理接口 301, 302
- 设置
 - EGL_GENERATORS_PLUGINDIR 307
- 输出类型 483, 484
- 向导 298
- EGL 命令文件 301
- EGL SDK 302
- EGLCMD 301, 436
- eglpPath 435
- EGLSDK 302, 446
- EJB 项目, 部署代码 307
- Java 包装器 273
- Java 包装器输出 615
- Java 输出 297, 614
- Java 选项 273
- 生成选项
 - Java 273
- 声明
 - 变量 49
 - 常量 49
- 使用声明 875
- 首选项
 - 模板 109
 - 文本 105
 - 源样式 108
 - EGL 105
 - EGL 编辑器 108
 - EGL 表单编辑器 160
 - EGL 表单编辑器选用板条目 156
 - EGL 调试器 106
 - EGL 至 EGL 迁移 102
 - SQL 检索 112
 - SQL 数据库连接 110
- 输出
 - 生成的类型 483, 484
 - Java 包装器生成 615
 - Java 生成 297, 614
- 输入表单 672
- 输入记录 672
- 数据部件
 - 串行记录 679
 - 带索引记录 488
 - 基本记录 345
 - 相关记录 676
 - dataItem 121, 431
 - dataTable 134, 432
 - MQ 记录 603
 - SQL 记录 683
- 数据初始化 430
- 数据代码, SQL 680
- 数据库连接首选项 110
- 数据库授权 425
- 数据转换 426
- 属性
 - 变长记录 673

属性 (续)

- 部件 59
- 程序部件 670
- 格式 61
- 验证 62
- 页面字段 624
- 字段 59
- 字段显示 60
- ConsoleField 401
- Java 运行时 315, 493
- MQ 记录 606
- SQL 字段 61
- 属性, 基本字段级别
 - action 628
 - align 629
 - byPassValidation 629
 - color 630
 - column 631
 - currency 632
 - currencySymbol 632
 - dateFormat 633
 - displayName 635
 - displayUse 635
 - fieldLen 636
 - fill 637
 - fillCharacter 637
 - help 637
 - highlight 638
 - inputRequired 638
 - inputRequiredMsgKey 638
 - intensity 639
 - isBoolean 639
 - isDecimalDigit 639
 - isHexDigit 640
 - isNullable 640
 - isReadOnly 641
 - lineWrap 642
 - lowerCase 642
 - masked 642
 - maxLength 643
 - minimumInput 643
 - minimumInputMsgKey 644
 - modified 644
 - needsSOSI 644
 - newWindow 645
 - numElementsItem 646
 - numericSeparator 646
 - outline 646
 - pattern 647
 - persistent 647
 - protect 648
 - selectFromListItem 649
 - selectType 649
 - sign 650
 - sqlDataCode 650
 - sqlVariableLen 651

属性, 基本字段级别 (续)

- timeFormat 652
- timeStampFormat 653
- typeChkMsgKey 654
- upperCase 654
- validationOrder 655
- validatorDataTable 655
- validatorDataTableMsgKey 656
- validatorFunction 656
- validatorFunctionMsgKey 657
- validValues 658
- validValuesMsgKey 659
- value 659
- zeroFormat 660
- 数字表达式 461
- 数组 68
 - 动态数组 68
 - 函数 69
 - appendAll() 69
 - appendElement() 69
 - getMaxSize() 70
 - getSize() 70
 - insertElement() 70
 - removeAll() 70
 - removeElement() 70
 - resizeAll() 71
 - resize() 70
 - setMaxSizes() 71
 - setMaxSize() 71
- 结构字段 71
- 双向语言文本转换 429

[W]

- 文本表达式 462
- 文本表单 145
- 文本应用程序
 - 分段 147
 - 启动 308
 - 已修正数据标记 148
 - formGroup 部件 141
- 文本, 首选项 105
- 文件
 - 程序属性 317
 - 创建 118, 267
 - 构建 13, 267
 - 结果 298
 - 链接属性 330, 598
 - 与记录类型相关联 673
 - 源 13, 118
 - 在“项目资源管理器”中删除 254
 - EGL 命令 439
 - J2EE 环境 324
 - Web Service 定义 13
- 文件和数据库系统字
 - recordName.resourceAssociation 786

文件夹, 创建 117

[X]

系统库

 DateTimeLib 723

系统限制 451

系统字

 Web 应用程序 734

显式 SQL 语句 237, 238, 239

相关记录部件 676

项目

 创建 116

 描述 13

 指定数据库选项 117

 EJB, 部署代码生成 307

 EJB, JNDI 名称 325

消息队列

 与 MQSeries 相关的 EGL 关键字 244

 远程 246

 MQ 记录属性 606

 MQ 选项记录 606

 MQSeries 支持 242

 MQSeries 直接调用 246

[Y]

验证属性 62

页面字段属性 624

一维数组 68

已修正数据标记 148

异常

 处理 86

 EGL 系统 86, 450

 I/O 错误值 490

 try 块 86

隐式 SQL 语句 236, 237, 238, 239

引用

 变量 53

 部件 20

 常量 53

引用兼容性 675

引用类型 168

用户界面 (UI) 部件

 编辑 151

 表单 142, 467

 页面字段属性 624

 formGroup 141, 464

用于生成的批处理接口 301, 302

用于 Java 的 EGL 运行时编码, 安装
318

游标, SQL 209

语法图 690

语句

 变量声明 80

语句 (续)

 常量声明 80

 赋值 80, 340

 关键字 80

 函数调用 80, 473

 null 80

 SQL 209

源文件

 编辑器

 对源代码进行注释 251

 创建 118

 描述 13

 内容辅助 118, 441

 在“项目资源管理器”中定位 253

 注释 251

 format 448

源样式, 首选项 108

运行单元 678

运行时环境, J2EE 设置 321

运算符

 优先顺序 613

 in 486

 isa 493

[Z]

在程序之间转移控制权 85

主变量, SQL 680

主构建描述符

 概述 270

 eglmaster.properties 448

 plugin.xml 463

注释 399

注释, 源代码 251

转换

 双向语言文本 429

 data 426

资源关联部件

 编辑 281

 除去 282

 关联元素 340

 描述 277

 添加 280

子串 688

自变量堆栈

 C 函数 395, 397

字典

 函数

 containsKey() 78

 getKeys() 78

 getValues() 78

 insertAll() 78

 removeAll() 79

 removeElement() 79

 size() 79

 描述 75

字典 (续)

 属性 76

字段

 结构 686

 属性 59

 属性, 页 624

 属性, SQL 61

 ConsoleField 401

 Menu 414

 MenuItem 415

 PresentationAttributes 417

 Prompt 419

 Window 420

字段显示属性 60

A

abs() 769

acos() 769

action

 基本字段级别属性 628

activateWindowByName() 696

activateWindow() 695

activeForm 695

activeWindow 695

add 语句 511

addReportData() 789

addReportParameter() 789

alias 与转移相关的元素属性 874

alias callLink 元素属性 372

align

 基本字段级别属性 629

ANY 34

appendAll() 69

appendElement() 69

arrayDictionary 部件

 描述 79

arrayIndex 851

asin() 770

asynchLink 元素

 描述 343

 package 344

 recordName 344

atan2() 770

atan() 770

attachBlobToFile() 761

attachBlobToTempFile() 761

attachClobToFile() 761

attachClobToTempFile() 762

B

beginDatabaseTransaction() 815

BIGINT 函数 389

BLOB 45

buildPlan 构建描述符选项 350
byPassValidation
 基本字段级别属性 629
bytes() 815

C

C 函数

调用 386, 390, 393
自变量堆栈 395, 397
DATE 391
DATETIME 391
DECIMAL 392
EGL 386
INTERVAL 391

C 数据类型 390

calculateChkDigitMod10() 816
calculateChkDigitMod11() 817
call 语句 513
callCmd() 818
callConversionTable 852
callLink 元素
 库 375
 描述 370
 alias 372
 conversionTable 372
 ctgKeyStore 373
 ctgKeyStorePassword 373
 ctgLocation 373
 ctgPort 374
 JavaWrapper 374
 linkType 375
 location 376
 luwControl 377
 package 378
 parmForm 378
 pgmName 379
 providerURL 379
 refreshScreen 380
 remoteBind 381
 remoteComType 381
 remotePgmType 383
 serverID 384
 type 385
cancelArrayDelete() 696
cancelArrayInsert() 696
case 语句 515
ceiling() 771
CHAR 35
characterAsInt() 796
CICSJ2C 调用设置 325
cicsj2cTimeout 构建描述符选项 351
clearActiveForm() 696
clearActiveWindow() 697
clearFieldsByName() 697
clearFields() 697

clearForm() 698
clearRequestAttr() 734
clearScreen() 721
clearSessionAttr() 734
clearWindowByName() 698
clearWindow() 698
clip() 797
CLOB 基本类型 44
close 语句 517
closeActiveWindow() 698
closeWindowByName() 699
closeWindow() 699
color
 基本字段级别属性 630
column
 基本字段级别属性 631
commentLevel 构建描述符选项 351
commentLine 700
commitOnConverse 845
commit() 818
compareNum() 771
compareStr() 797
concatenateWithSeparator() 799
concatenate() 798
conditionAsInt() 819
ConnectionFactory, CICSJ2C 325
connectionService() 839
connect() 820
ConsoleField
 属性 401
 字段 401
ConsoleForm
 部件属性 413
ConsoleLib
 activateWindowByName() 696
 activateWindow() 695
 activeForm 695
 activeWindow 695
 cancelArrayDelete() 696
 cancelArrayInsert() 696
 clearActiveForm() 696
 clearActiveWindow 697
 clearFieldsByName() 697
 clearFields() 697
 clearForm() 698
 clearWindowByName() 698
 clearWindow() 698
 closeActiveWindow() 698
 closeWindowByName() 699
 closeWindow() 699
 commentLine 700
 currentArrayCount() 700
 currentArrayDataLine() 700
 currentArrayScreenLine() 700
 currentDisplayAttrs 701
 currentRowAttrs 701

ConsoleLib (续)

cursorWrap 701
defaultDisplayAttributes 701
defaultInputAttributes 702
deferInterrupt 702
deferQuit 702
definedFieldOrder 702
displayAtLine() 703
displayAtPosition() 703
displayError() 703
displayFieldsByName() 704
displayFields() 704
displayFormByName() 705
displayForm() 704
displayLineMode() 705
displayMessage() 705
drawBoxWithColor() 706
drawBox() 705
errorLine 707
errorWindow 707
errorWindowVisible 707
formLine 707
getKeyCode() 708
getKeyName() 708
getKey() 707
gotoFieldByName() 709
gotoField() 708
gotoMenuItemByName() 709
gotoMenuItem() 709
hideAllMenuItems() 709
hideErrorWindow() 710
hideMenuItemByName() 710
hideMenuItem() 710
interruptRequested 710
isCurrentFieldByName() 711
isCurrentField() 711
isFieldModifiedByName() 712
isFieldModified() 711
key_accept 712
key_deleteLine 712
key_help 713
key_insertLine 713
key_interrupt 713
key_pageDown 713
key_pageUp 713
key_quit 714
lastKeyTyped() 714
menuLine 714
messageLine 714
messageResource 714
nextField() 715
openWindowByName() 715
openWindowWithFormByName() 716
openWindowWithForm() 716
openWindow() 715
previousField() 716

ConsoleLib (续)
 promptLine 717
 promptLineMode() 717
 quitRequested 717
 screen 717
 scrollDownLines() 717
 scrollDownPage() 718
 scrollUpLines() 718
 scrollUpPage() 718
 setArrayLine() 718
 setCurrentArrayCount() 719
 showAllMenuItems() 719
 showHelp() 719
 showMenuItemByName() 720
 showMenuItem() 719
 sqlInterrupt 720
ConsoleUI 164
 概述 165
 OpenUI 语句 565
ConsoleUI 概述 163
ConsoleUI 屏幕选项
 UNIX 用户 169
containsKey() 78
continue 语句 519
converse 语句 519
ConverseLib
 clearScreen() 721
 displayMsgNum() 721
 fieldInputLength() 722
 pageEject() 722
 validationFailed() 722
ConverseVar
 commitOnConverse 845
 eventKey 845
 printerAssociation 846
 segmentedMode 848
 validationMsgNum 849
conversionTable callLink 元素属性 372
convert() 822
copyStr() 800
cosh() 773
cos() 772
csouidpwd.properties 343
ctgKeyStore callLink 元素属性 373
ctgKeyStorePassword callLink 元素属性 373
ctgLocation callLink 元素属性 373
ctgPort callLink 元素属性 374
currency
 基本字段级别属性 632
currencySymbol
 基本字段级别属性 632
currencySymbol 构建描述符选项 351
currentArrayCount() 700
currentArrayDataLine() 700
currentArrayScreenLine() 700

currentDate() 724
currentDisplayAttrs 701
currentFormattedGregorianDate 863
currentFormattedJulianDate 864
currentFormattedTime 865
currentGregorianDate 865
currentJulianDate 866
currentRowAttrs 701
currentShortGregorianDate 866
currentShortJulianDate 867
currentTimeStamp() 725
currentTime() 725
curses 库, UNIX 319
cursorWrap 701

D

dataItem 部件
 创建 121
 描述 121
 EGL 源格式 431
dataTable 部件
 创建 134
 描述 134
 EGL 源格式 432
DATE 38
DATE 函数 391
dateFormat
 基本字段级别属性 633
dateOf() 726
DATETIME 函数 391
DateTimeLib 723
 currentDate() 724
 currentTimeStamp() 725
 currentTime() 725
 dateOf() 726
 dateValueFromGregorian() 726
 dateValueFromJulian() 727
 dateValue() 726
 dayOf() 727
 extend() 728
 intervalValueWithPattern() 729
 intervalValue() 728
 mdy() 729
 monthOf() 730
 timeOf() 730
 timeStampFrom() 731
 timeStampValueWithPattern() 732
 timeStampValue() 731
 timeValue() 732
 weekdayOf() 733
 yearOf() 733
dateValueFromGregorian() 726
dateValueFromJulian() 727
dateValue() 726
dayOf() 727

DBCHAR 35
dbms 构建描述符选项 352
DECIMAL 46
DECIMAL 函数 392
decimalSymbol 构建描述符选项 352
defaultDateFormat 801
defaultDisplayAttributes 701
defaultInputAttributes 702
defaultMoneyFormat 801
defaultNumericFormat 802
defaultTimeFormat 802
defaultTimestampFormat 802
deferInterrupt 702
deferQuit 702
defineDatabaseAlias() 823
definedFieldOrder 702
delete 语句 520
destDirectory 构建描述符选项 352
destHost 构建描述符选项 353
destPassword 构建描述符选项 353
destPort 构建描述符选项 353
destUserID 构建描述符选项 354
disconnectAll() 825
disconnect() 825
display 语句 522
displayAtLine() 703
displayAtPosition() 703
displayError() 703
displayFieldsByName() 704
displayFields() 704
displayFormByName() 705
displayForm() 704
displayLineMode() 705
displayMessage() 705
displayMsgNum() 721
displayName
 基本字段级别属性 635
displayUse
 基本字段级别属性 635
drawBoxWithColor() 706
drawBox() 705

E

ear 文件, 消除重复的 jar 文件 322
EGL 保留字 444
EGL 编辑器
 概述 441
 内容辅助 441
 首选项 108
EGL 表单编辑器
 概述 150
 首选项 160
 显示选项 160
EGL 调试器
 查看变量 266

- EGL 调试器 (续)
 - 创建启动配置 261
 - 创建侦听器启动配置 262
 - 从生成的代码中调用 255
 - 单步执行程序 265
 - 断点 264
 - 概述 255
 - 构建描述符 255
 - 建议 255
 - 命令 255
 - 启动程序 260
 - 启动服务器 263
 - 启动 Web 会话 263
 - 设置首选项 106
 - 系统类型首选项 255
 - 准备服务器 262
 - 字符编码选项 107
 - call 语句 255
 - SQL 数据库访问 255
- EGL 概述 1
- EGL 构建路径
 - 编辑 290
 - 概述 435
- EGL 构建文件格式 346
- EGL 基本类型 390
- EGL 命令文件 439
- EGL 属性
 - 概述 59
- EGL 源格式 448
- EGL 中的新增内容 1
- EGL 中的 JDBC 驱动程序需求 510
- EGL 6.0 中的新增内容 4
- EGL 6.0 iFix 中的新增内容 3
- EGL Java 运行时错误代码 879
- EGL Java 运行时的消息定制 602
- EGL SDK (EGL Software Development Kit) 302
- EGLCMD 301, 436
- eglmaster.properties 448
- eglpsh 435
- EGLSDK 446
- EGL_GENERATORS_PLUGINDIR 变量 307
- EJB 会话
 - 描述 286
 - 组件 286
- EJB 项目
 - 部署代码生成 307
 - 设置 JNDI 名称 325
- eliminateSystemDependentCode 构建描述符选项 354
- enableJavaWrapperGen 构建描述符选项 355
- errorCode 853
- errorLine 707
- errorLog() 825

- errorWindow 707
- errorWindowVisible 707
- eventKey 845
- execute 语句 522
- exit 语句 526
- exportReport() 790
- exp() 773
- extend() 728
- externallyDefined transferToTransaction 元素属性 874

F

- fieldInputLength() 722
- fieldLen
 - 基本字段级别属性 636
- fill
 - 基本字段级别属性 637
- fillCharacter
 - 基本字段级别属性 637
- fillReport() 791
- findStr() 803
- floatingAssign() 773
- floatingDifference() 774
- floatingMod() 774
- floatingProduct() 775
- floatingQuotient() 775
- floatingSum() 776
- floor() 777
- for 语句 528
- forEach 语句 530
- formatDate() 804
- formatNumber() 805
- formatTimeStamp() 806
- formatTime() 805
- formConversionTable 854
- formGroup 部件
 - 编辑 151
 - 创建 141
 - 描述 141
 - 使用声明 877
 - EGL 源格式 464
 - pfKeyEquate 属性 625
- formLine 707
- forward 语句 531
- freeBlob() 762
- freeClob() 762
- freeSQL 语句 532
- frexp() 777

G

- genDataTables 构建描述符选项 355
- genDirectory 构建描述符选项 355
- genFormGroup 构建描述符选项 356

- genHelpFormGroup 构建描述符选项 356
- genProject 构建描述符选项 357
- genProperties 构建描述符选项 358
- get 语句 533
- get absolute 语句 538
- get current 语句 540
- get first 语句 541
- get last 语句 543
- get next 语句 544
- get previous 语句 549
- get relative 语句 552
- getBlobLen() 762
- getClobLen() 763
- getCmdLineArgCount() 826
- getCmdLineArg() 826
- getFieldValue() 792
- getField() 743
- getKeyCode() 708
- getKeyName() 708
- getKeys() 78
- getKey() 707
- getMaxSize() 70
- getMessage() 827
- getNextToken() 807
- getProperty() 828
- getReportData() 792
- getReportParameter() 793
- getReportVariableValue() 793
- getRequestAttr() 735
- getSessionAttr() 735
- getSize() 70
- getStrFromClob() 763
- getSubStrFromClob() 764
- getVAGSysType() 842
- getValues() 78
- goTo 语句 554
- gotoFieldByName() 709
- gotoField() 708
- gotoMenuItemByName() 709
- gotoMenuItem() 709

H

- handleHardIOErrors 867
- handleOverflow 868
- handleSysLibraryErrors 869
- help
 - 基本字段级别属性 637
- HEX 36
- hideAllMenuItems() 709
- hideErrorWindow() 710
- hideMenuItemByName() 710
- hideMenuItem() 710
- highlight
 - 基本字段级别属性 638

I

- I4GL 数据类型 390
- if, else 语句 555
- import 30
- in 运算符 486
- Informix
 - 特殊注意事项 230
- inputRequired
 - 基本字段级别属性 638
- inputRequiredMsgKey
 - 基本字段级别属性 638
- insertAll() 78
- insertElement() 70
- integerAsChar() 809
- intensity
 - 基本字段级别属性 639
- interruptRequested 710
- INTERVAL 38
- INTERVAL 函数 391
- intervalValueWithPattern() 729
- intervalValue() 728
- invoke() 745
- isa 运算符 493
- isBoolean
 - 基本字段级别属性 639
- isCurrentFieldByName() 711
- isCurrentField() 711
- isDecimalDigit
 - 基本字段级别属性 639
- isFieldModifiedByName() 712
- isFieldModified() 711
- isHexDigit
 - 基本字段级别属性 640
- isNullable
 - 基本字段级别属性 640
- isNull() 748
- isObjId() 749
- isReadOnly
 - 基本字段级别属性 641
- I/O 错误值 490

J

- J2EE 部署设置
 - 描述符值 322, 324
 - 运行时环境 321
 - ConnectionFactory, CICSJ2C 325
 - JDBC 连接 328
 - TCP/IP 侦听器 320, 326
- J2EE 构建描述符选项 360
- J2EE 环境文件
 - 更新 323
 - 描述 324
- J2EE JDBC 连接 328

J2EELib

- clearRequestAttr() 734
- clearSessionAttr() 734
- getRequestAttr() 735
- getSessionAttr() 735
- setRequestAttr() 736
- setSessionAttr() 736

jar 文件, 运行时

- 提供访问 330
- 消除 ear 文件中的重复内容 322

Java 包装器

- 别名 610
- 类 502
- 描述 274
- 生成 273
- 生成输出 615
- 使用 8

Java 别名 610

Java 访问函数 737

Java 运行时属性 315, 493

JavaLib

- getField() 743
- invoke() 745
- isNull() 748
- isObjId() 749
- qualifiedTypeName() 750
- removeAll() 752
- remove() 751
- setField() 752
- storeCopy() 756
- storeField() 757
- storeNew() 758
- store() 754

JavaServer Faces 180

JavaWrapper callLink 元素属性 374

JDBC 连接

- 标准 240
- J2EE 328

JNDI 名称, 为 EJB 项目设置 325

JSP 175

K

- key_accept 712
- key_deleteLine 712
- key_help 713
- key_insertLine 713
- key_interrupt 713
- key_pageDown 713
- key_pageUp 713
- key_quit 714

L

- lastKeyTyped() 714
- Ldexp() 777
- library callLink 元素属性 375
- like 597
- lineWrap
 - 基本字段级别属性 642
- linkage 构建描述符选项 360
- linkType callLink 元素属性 375
- loadBlobFromFile() 764
- loadClobFromFile() 764
- loadTable() 828
- LobLib 760
 - attachBlobToFile() 761
 - attachBlobToTempFile() 761
 - attachClobToFile() 761
 - attachClobToTempFile() 762
 - freeBlob() 762
 - freeClob() 762
 - getBlobLen() 762
 - getClobLen() 763
 - getStrFromClob() 763
 - getSubStrFromClob() 764
 - loadBlobFromFile() 764
 - loadClobFromFile() 764
 - setClobFromStringAtPosition() 765
 - setClobFromString() 765
 - truncateBlob() 766
 - truncateClob() 766
 - updateBlobToFile() 766
 - updateClobToFile() 767
- location callLink 元素属性 376
- log10() 778
- log() 778
- lowerCase
 - 基本字段级别属性 642
- lowerCase() 809
- luwControl callLink 元素属性 377

M

- masked
 - 基本字段级别属性 642
- matches 601
- MathLib
 - abs() 769
 - acos() 769
 - asin() 770
 - atan2() 770
 - atan() 770
 - ceiling() 771
 - compareNum() 771
 - cosh() 773
 - cos() 772
 - exp() 773

MathLib (续)
floatingAssign() 773
floatingDifference() 774
floatingMod() 774
floatingProduct() 775
floatingQuotient() 775
floatingSum() 776
floor() 777
frexp() 777
Ldexp() 777
log10() 778
log() 778
maximum() 779
minimum() 779
modf() 780
pow() 780
precision() 781
round() 781
sinh() 783
sin() 782
sqrt() 783
stringAsDecimal() 784
stringAsFloat() 784
stringAsInt() 785
tanh() 786
tan() 785
maximumSize() 829
maximum() 779
maxLen
 基本字段级别属性 643
MBCHAR 36
mdy() 729
Menu
 字段 414
MenuItem
 字段 415
menuLine 714
messageLine 714
messageResource 714
minimumInput
 基本字段级别属性 643
minimumInputMsgKey
 基本字段级别属性 644
minimum() 779
modf() 780
modified
 基本字段级别属性 644
monthOf() 730
move 语句 556
MQ 记录部件
 属性 606
 选项记录 606
 EGL 源格式 603
mqConditionCode 869
MQSeries
 相关的 EGL 关键字 244

MQSeries (续)
 支持 242
 直接调用 246
MQ 记录属性 606
MQ 选项记录 606

N

needsSOSI
 基本字段级别属性 644
newWindow
 基本字段级别属性 645
nextBuildDescriptor 构建描述符选项 360
nextField() 715
null 209
NUM 47
NUMC 48
numElementsItem
 基本字段级别属性 646
numericSeparator
 基本字段级别属性 646

O

open 语句 561
OpenUI 语句 565
openWindowByName() 715
openWindowWithFormByName() 716
openWindowWithForm() 716
openWindow() 715
outline
 基本字段级别属性 646
output
 重建项目菜单选项 295
 构建 296
 构建项目菜单选项 295
 全部重建菜单选项 295
overflowIndicator 854

P

PACF 48
package asynchLink 元素属性 344
package callLink 元素属性 378
Page Designer
 绑定 175
 单选组件 186
 多选组件 187
 复选框组件 185
 基本类型 181
 记录 182
 命令组件 183
 输出组件 184
 输入组件 184
 支持 175

Page Designer (续)
 “快速编辑”视图, 页面处理程序代码 184
pageEject() 722
PageHandler 部件
 描述 177
pageHandler 部件
 绑定单选组件 186
 绑定多选组件 187
 绑定复选框组件 185
 绑定命令组件 183
 绑定输出组件 184
 绑定输入组件 184
 创建 175
 使用声明 878
 EGL 源格式 619
parmForm callLink 元素属性 378
pattern
 基本字段级别属性 647
persistent
 基本字段级别属性 647
pfKeyEquate 属性 625
pgmName callLink 元素属性 379
plugin.xml 463
pow() 780
precision() 781
prep 构建描述符选项 361
prepare 语句 574
PresentationAttributes
 字段 417
previousField() 716
print 语句 576
printerAssociation 846
Prompt
 字段 419
promptLine 717
promptLineMode() 717
protect
 基本字段级别属性 648
providerURL callLink 元素属性 379

Q

qualifiedTypeName() 750
queryCurrentDatabase() 829
quitRequested 717

R

recordName asynchLink 元素属性 344
refreshScreen callLink 元素属性 380
remoteBind callLink 元素属性 381
remoteComType callLink 元素属性 381
remotePgmType callLink 元素属性 383
removeAll() 70, 79, 752

removeElement() 70, 79
remove() 751
replace 语句 576
ReportLib
 addReportData() 789
 addReportParameter() 789
 exportReport() 790
 fillReport() 791
 getFieldValue() 792
 getReportData() 792
 getReportParameter() 793
 getReportVariableValue() 793
 resetReportParameters() 793
 setReportVariableValue() 794
resetReportParameters() 793
resizeAll() 71
resize() 70
resourceAssociations 构建描述符选项 361
resultSetID 678
return 语句 579
returnCode 855
rollback() 830
round() 781

S

screen 717
scrollDownLines() 717
scrollDownPage() 718
scrollUpLines() 718
scrollUpPage() 718
segmentedMode 848
selectFromListItem
 基本字段级别属性 649
selectType
 基本字段级别属性 649
serverID callLink 元素属性 384
sessionBeanID 构建描述符选项 362
sessionId 856
set 语句 579
setArrayLine() 718
setBlankTerminator() 810
setClobFromStringAtPosition() 765
setClobFromString() 765
setCurrentArrayCount() 719
setCurrentDatabase() 830
setError() 831
setField() 752
setLocale() 832
setMaxSizes() 71
setMaxSize() 71
setNullTerminator() 810
setRemoteUser() 832
setReportVariableValue() 794
setRequestAttr() 736
setSessionAttr() 736
setSubStr() 810
set-value 块 62
show 语句 588
showAllMenuItems() 719
showHelp() 719
showMenuItemByName() 720
showMenuItem() 719
sign
 基本字段级别属性 650
sinh() 783
sin() 782
size() 79, 833
Software Development Kit, EGL (EGL SDK) 302
spaces() 811
SQL
 创建 dataItem 部件 232
 动态语句 219
 构造 PREPARE 语句 237
 记录部件 209
 记录内部结构 682
 检索功能 209, 230, 231
 检索首选项 112
 结果集处理 209, 678
 缺省数据库 230
 示例 219
 数据代码 680
 数据库连接首选项 110
 数据库授权 425
 显式语句 209, 237, 238, 239
 隐式语句 209, 236, 237, 238, 239
 游标 209
 支持 209, 230
 主变量 680
 EGL 语句 209
 null 209
 SQL 记录部件 683
 SQL 字段属性 61
 sqlca 857
 sqlcode 858
 sqlDataCode
 基本字段级别属性 650
 sqlDB 构建描述符选项 364
 sqlerrd 870
 sqlerrmc 871
 sqlID 构建描述符选项 364
 sqlInterrupt 720
 sqlIsolationLevel 872
 sqlJDBCdriverClass 构建描述符选项 365
 sqlJNDIName 构建描述符选项 365
 sqlPassword 构建描述符选项 366
 sqlState 858
 sqlValidationConnectionURL 构建描述符选项 366
 sqlVariableLen
 基本字段级别属性 651
sqlWarn 872
sqrt() 783
startCmd() 834
startLog() 835
startTransaction() 835
storeCopy() 756
storeField() 757
storeNew() 758
store() 754
STRING 37
stringAsDecimal() 784
stringAsFloat() 784
stringAsInt() 785
strlen() 812
StrLib
 characterAsInt() 796
 clip() 797
 compareStr() 797
 concatenateWithSeparator() 799
 concatenate() 798
 copyStr() 800
 defaultDateFormat 801
 defaultMoneyFormat 801
 defaultNumericFormat 802
 defaultTimeFormat 802
 defaultTimestampFormat 802
 findStr() 803
 formatDate() 804
 formatNumber() 805
 formatTimeStamp() 806
 formatTime() 805
 getNextToken() 807
 integerAsChar() 809
 lowerCase() 809
 setBlankTerminator() 810
 setNullTerminator() 810
 setSubStr() 810
 spaces() 811
 strlen() 812
 textLen() 812
 upperCase() 812
SysLib
 beginDatabaseTransaction() 815
 bytes() 815
 calculateChkDigitMod10() 816
 calculateChkDigitMod11() 817
 callCmd() 818
 commit() 818
 conditionAsInt() 819
 connect() 820
 convert() 822
 defineDatabaseAlias() 823
 disconnectAll() 825
 disconnect() 825
 errorLog() 825
 getCmdLineArgCount() 826

SysLib (续)
getCmdLineArg() 826
getMessage() 827
getProperty() 828
loadTable() 828
maximumSize() 829
queryCurrentDatabase() 829
rollback() 830
setCurrentDatabase() 830
setError() 831
setLocale() 832
setRemoteUser() 832
size() 833
startCmd() 834
startLog() 835
startTransaction() 835
unloadTable() 836
verifyChkDigitMod10() 836
verifyChkDigitMod11() 837
wait() 839

system 构建描述符选项 367
systemType 859

SysVar
arrayIndex 851
callConversionTable 852
errorCode 853
formConversionTable 854
overflowIndicator 854
returnCode 855
sessionID 856
sqlca 857
sqlcode 858
sqlState 858
systemType 859
terminalID 860
transactionID 860
transferName 861
userID 861

T

tanh() 786
tan() 785
targetNLS 构建描述符选项 368
TCP/IP 侦听器 320, 326
terminalID 860
textLen() 812
textUI 程序部件 668
TIME 40
timeFormat
基本字段级别属性 652
timeOf() 730
TIMESTAMP 40
timeStampFormat
基本字段级别属性 653
timeStampFrom() 731

timeStampValueWithPattern() 732
timeStampValue() 731
timeValue() 732
transactionID 860
transfer 语句 589
transferName 861
transferToProgram 元素
alias 874
transferToTransaction 元素
描述 873
alias 874
externallyDefined 874
truncateBlob() 766
truncateClob() 766
try 语句 590
type callLink 元素属性 385
typeChkMsgKey
基本字段级别属性 654
typedef 25

U

UNICODE 37
UNIX 用户
ConsoleUI 屏幕选项 169
UNIX curses 库 319
unloadTable() 836
updateBlobToFile() 766
updateClobToFile() 767
upperCase
基本字段级别属性 654
upperCase() 812
userID 861

V

VAGCompatibility 构建描述符选项 368
validateSQLStatements 构建描述符选项
369
validationFailed() 722
validationMsgNum 849
validationOrder
基本字段级别属性 655
validatorDataTable
基本字段级别属性 655
validatorDataTableMsgKey
基本字段级别属性 656
validatorFunction
基本字段级别属性 656
validatorFunctionMsgKey
基本字段级别属性 657
validValues
基本字段级别属性 658
validValuesMsgKey
基本字段级别属性 659

value
基本字段级别属性 659
verifyChkDigitMod10() 836
verifyChkDigitMod11() 837
VGLib
connectionService() 839
getVAGSysType() 842
VGVar
currentFormattedGregorianCalendar 863
currentFormattedJulianDate 864
currentFormattedTime 865
currentGregorianCalendar 865
currentJulianDate 866
currentShortGregorianCalendar 866
currentShortJulianDate 867
handleHardIOErrors 867
handleOverflow 868
handleSysLibraryErrors 869
mqConditionCode 869
sqlerrd 870
sqlerrmc 871
sqlIsolationLevel 872
sqlWarn 872
VisualAge Generator
迁移自 11
EGL 兼容性 400
VSAM
访问先决条件 241
系统名称 241
支持 241

W

wait() 839
Web 应用程序
支持 171
Page Designer 175
Web 应用程序系统字 734
Web Service 定义文件 13
weekdayOf() 733
while 语句 591
Window
字段 420

X

XML 报告设计文档
概述 189
其中的数据类型 196
添加至包 198

Y

yearOf() 733

Z

zeroFormat

基本字段级别属性 660

[特别字符]

“快速编辑”视图

页面处理程序代码 184



程序号: 5724-J19

中国印刷