WebSphere MQ Everyplace V2.0.2

Contenido

Desarrollo de una aplicación básica

Este tema contiene la información necesaria para crear una aplicación de MQe sencilla. En él se presenta el kit de herramientas de desarrollo de MQe y se explica qué es necesario para configurar el entorno de desarrollo. Una guía proporciona las instrucciones paso a paso sobre cómo crear una aplicación de MQe sencilla y cómo verificar si funciona.

También se describe un ejemplo de aplicación denominado HelloWorld. Esta aplicación sencilla demuestra cómo utilizar algunas de las funciones de MQe. Finalmente, se presentan algunas de las herramientas que se pueden utilizar para desarrollar y administrar aplicaciones de MQe.

Introducción al kit de desarrollo de WebSphere MQe

En este tema se presenta el kit de desarrollo de WebSphere MQe, que es un entorno de desarrollo para escribir aplicaciones de mensajería y de colas basadas en Java y C. Para obtener información sobre la disponibilidad de los kits de desarrollo que no sean Java y C, consulte el sitio web de WebSphere MQ en:

`http://www.ibm.com/software/ts/mqseries`

La parte de código del Java Development Kit se presenta en dos apartados:

Clases base de WebSphere MQ Everyplace
: Conjunto de clases Java que proporcionan todas las funciones necesarias para crear aplicaciones de mensajería y colas.

Ejemplos
: Clases y código fuente en Java que muestran cómo utilizar muchas funciones de MQe.

La parte de código del kit de desarrollo en C también se presenta en dos apartados:

Funciones base de WebSphere MQ Everyplace
: Código C que proporciona todas las funciones necesarias para crear aplicaciones de mensajería y de gestión de colas.

Ejemplos
: Código fuente en C que muestra cómo utilizar muchas funciones de MQe.

Configuración del entorno de desarrollo

En este tema se proporciona información sobre la configuración del entorno de desarrollo para Java y C.

Desarrollo en Java

Para desarrollar programas en Java con el kit de desarrollo de MQe, debe configurar el entorno Java de la manera siguiente:

- Establezca la variable *CLASSPATH* de manera que el JDK (Java Development Kit) pueda localizar las clases de MQe.

  Windows

  En un entorno Windows, mediante un JDK estándar, puede utilizar lo siguiente:

  ```
  Set CLASSPATH=<MQeInstallDir>\Java;%CLASSPATH%
  ```

  UNIX En un entorno UNIX, puede utilizar lo siguiente:

```
CLASSPATH=<MQeInstallDir>/Java:$CLASSPATH

export CLASSPATH
```

Puede utilizar muchos entornos de desarrollo en Java y entornos de ejecución en Java distintos con MQe. La configuración del sistema para el entorno de desarrollo y de ejecución depende del entorno que se utilice. MQe incluye un archivo que muestra cómo configurar un entorno de desarrollo para distintos Java Development Kits. En sistemas Windows, se trata de un archivo de proceso por lotes denominado JavaEnv.bat, para sistemas UNIX, es un script de shell denominado JavaEnv. Para utilizar este archivo, copie el archivo y modifique la copia para que coincida con el entorno de la máquina en el que desea utilizarlo.

Un conjunto de archivos de proceso por lotes y scripts de shell que ejecutan algunos de los ejemplos de MQe utilizan el archivo de entorno que se ha descrito anteriormente. Si desea utilizar dichos archivos de ejemplo de proceso por lotes, deberá modificar el archivo de entorno del modo siguiente:

- Establezca la variable de entorno *JDK* en el directorio base de JDK.
- Establezca la variable de entorno *JavaCmd* en el mandato que se utiliza para ejecutar las aplicaciones en Java.
- Si se instalan las clases de MQ para Java, establezca la variable de entorno *MQDIR* en el directorio base de las clases de MQ para Java.

Nota: las versiones personalizadas de JavaEnv.bat o JavaEnv pueden sobrescribirse si reinstala MQe.

Cuando invoque JavaEnv.bat en Windows debe pasar un parámetro que determine el tipo de Java Development Kit que se debe utilizar.

Los valores posibles son:

Sun - Sun

JB - Borland JBuilder

MS - Microsoft

IBM - IBM

Nota: Estos parámetros son sensibles a las mayúsculas y minúsculas y deben escribirse exactamente tal como se muestran.

Si no pasa un parámetro, el valor por omisión es IBM.

El script de shell JavaEnv en UNIX no utiliza el parámetro correspondiente.

En Windows, por omisión, debe ejecutar JavaEnv.bat desde el directorio <MQeInstallDir>\java\demo\Windows. En UNIX, por omisión, debe ejecutar JavaEnv desde el directorio <MQeInstallDir>/Java/demo/UNIX. Ambos archivos pueden modificarse para que se puedan ejecutar desde otros directorios o utilizar otros Java Development Kits.

Entorno J2ME

Existen dos entornos J2ME distintos:

CDC (Connected Device Configuration) y Profile
: Un ejemplo es Foundation + Applications en el entorno de CDC, que se puede desarrollar de forma eficaz como una aplicación J2SE (Java 2 Platform Standard Edition) normal. El único cambio necesario es la modificación de la opción bootclasspath para señalar el archivo de clase zip o jar CDC correspondiente.

Nota: Es posible que la opción 'bootclasspath' no esté disponible en todas las JVM (Java Virtual Machine).

CLDC (Connected Limited Device Configuration) y MIDP (Mobile Information Device Profile)
Las aplicaciones desarrolladas para MIDP también se pueden compilar mediante una JVM J2SE normal (de nuevo, utilizando la bootclasspath para señalar la biblioteca de clases Midp necesaria), pero habitualmente se tienen que ejecutar en un emulador Midp. Por lo tanto, se recomienda desarrollar la aplicación utilizando uno de los kits de herramientas MIDP disponibles en la web. MQe proporciona un archivo .jar MIDP que se debe utilizar en este entorno. El archivo `MQeMidp.jar` se encuentra en el directorio <directorio_instalación_MQe>\Java\Jars.

Desarrollo en C

Para desarrollar programas en C con el kit de desarrollo de MQe, necesita las herramientas siguientes:

Microsoft eMbedded Visual C++ (EVC) Versión 3.0.
Se incluye en Microsoft eMbedded Visual Tools 3.0, que está disponible como descarga gratuita en la página web de Microsoft:

`http://msdn.microsoft.com/mobile/`

Deberá utilizar la versión 3.0, puesto que la versión 4.0 no da soporte a PocketPC.

Un SDK (Software Developers' Kit) para la plataforma que haya elegido
Microsoft eMbedded Visual Tools 3.0 incluye un SDK para PocketPC 2000. También puede descargar un SDK para PocketPC 2002 de Microsoft:

`http://msdn.microsoft.com/mobile/`

Enlaces de C

Para la base de código de enlaces C, consulte la consulta de programación de enlaces C.

C nativo

Para obtener información general, consulte la Consulta de programación en C, en concreto la página *Información de compilación*. No obstante, esa página ahora ha quedado un poco desfasada y este tema proporciona una actualización.

En el caso de la base de código en C nativa, se proporciona soporte para cuatro plataformas:

- PocketPC2000
- PocketPC2002
- PocketPC2003
- Windows de 32 bits.

En el caso de PocketPC, se proporcionan binarios tanto para el dispositivo como para el emulador que está disponible en el entorno de desarrollo integrado de Microsoft eMbedded Visual C++. Los archivos binarios que se proporcionan para los dispositivos están compilados para procesadores ARM.

Archivos binarios

La raíz de los archivos binarios, así como la documentación y los ejemplos, es el directorio C que depende del directorio que haya elegido para instalar MQe.

En el directorio C, los archivos se encuentran en las siguientes ubicaciones:

PocketPC2000

ARM

DLL C\PocketPc2000\arm\bin

LIB C\PocketPc2000\arm\lib

Emulador

DLL C\PocketPc2000\x86emulator\bin

LIB C\PocketPc2000\x86emulator\lib

PocketPC2002

ARM

DLL C\PocketPc2002\arm\bin

LIB C\PocketPc2002\arm\lib

Emulador

DLL C\PocketPc2002\x86emulator\bin

LIB C\PocketPc2002\x86emulator\lib

PocketPC2003

ARM

DLL C\PocketPc2003\arm\bin

LIB C\PocketPc2003\arm\lib

Emulador

DLL C\PocketPc2003\x86emulator\bin

LIB C\PocketPc2003\x86emulator\lib

Windows de 32 bits

DLL C\Win32\Native\bin

LIB C\Win32\Native\lib

Archivos de cabecera

Los archivos de cabecera son comunes para todas las plataformas nativas y los puede localizar en el directorio include que depende del directorio de instalación.

MQe_API.h

Éste es el archivo de cabecera ″raíz″. Si se incluye, todos los archivos de cabecera relevantes se incluirán de forma automática.

Para garantizar que se incluyan los archivos y las definiciones correctas, debe indicar que está ejecutando la base de código nativa de la manera siguiente:

```
#define   NATIVE                  // o especifíquelo como una opción para el compilador

#include  <published/MQe_API.h>
```

Enlazar

Tiene que enlazar con las dos bibliotecas siguientes:

HMQ_nativeAPI.lib
// biblioteca de API

HMQ_nativeCnst.lib
// biblioteca de MQeString constante estática

Debe incluir estos dos archivos. A continuación, un enlazador de optimización elimina los enlaces con las funciones y constantes que no haya utilizado.

Las otras bibliotecas de MQe se enlazan de forma estática y dinámica con la biblioteca de API principal y se incluyen según sea necesario.

Utilización de Visual C++ incorporado

Se pueden compilar aplicaciones utilizando el IDE (Integrated Development Environment) de EVC u opcionalmente desde la línea de mandatos. Sin embargo, se debe tener en consideración lo que se indica a continuación:

- Establezca la ″Configuración WCE activa″ apropiada, utilizando la barra de herramientas de configuración de WCE. Para realizar esta tarea, debajo de la opción **Sistema operativo de destino** seleccione PocketPC o PocketPC 2002. Asimismo, debajo de Procesador de destino, seleccione uno de los siguientes:
  - Win32 (WCE x86em) Debug
  - Win32 (WCE x86em) Release
  - Win32 (WCE ARM) Debug
  - Win32 (WCE ARM) Release

  Nota: Puede que algunas de las opciones de Procesador de destino o **Sistema operativo de destino** no estén disponibles, en función de los SDK que haya instalado.
- Incluya los archivos de cabecera para la base de código en C nativa. Las dos versiones de PocketPC y los enlaces C comparten estos archivos. La ubicación del archivo de cabecera se encuentra en el directorio de instalación, debajo del mandato `include`. Si incluye el archivo de cabecera de raíz, MQe_API.h, incluirá todas las funciones que pueda necesitar. Puesto que los archivos de cabecera se comparten, tendrá que definir la versión que utiliza de la base de código, tal como se muestra en el siguiente ejemplo:

```
#define   NATIVE

#define MQE_PLATFORM  PLATFORM_WINCE



/*Como alternativa, debe añadir esto a las definiciones de preprocesador

en el diálogo de valores del proyecto. Añada lo siguiente al inicio

de la lista */

NATIVE,MQE_PLATFORM=PLATFORM_WINCE



#include <published\MQe_API.h>
```

- Incluya una entrada para el directorio de inclusión de MQe de nivel superior en ″Directorios de inclusión adicionales″. Esto variará en función de la ubicación en la que instale el producto.
- Inserte los siguientes nombres de archivos .lib en el diálogo ″Configuración del proyecto″, debajo de **Vincular** → **Entrada**:
  - HMQ_nativeAPI.lib
  - HMQ_nativeCnst.lib

  Nota: Existen variaciones de estos archivos para cada release al que se da soporte, por ejemplo uno para PocketPC 2000 ARM, y otro para PocketPC 2000 x86em, y así sucesivamente. Para asegurarse de que utiliza la versión correcta, especifique el nombre de archivo completo de cada versión de compilación de destino.

Se recomienda efectuar el desarrollo de aplicaciones utilizando el emulador de PocketPC o de PocketPC2002, puesto que normalmente proporcionan un entorno de depuración y una compilación más

rápida. No obstante, los emuladores actuales son emuladores API, lo cual significa que no emulan hardware ARM. Estos emuladores emulan llamadas API de PocketPC, pero el código sigue siendo x86, que se ejecuta en una máquina virtual x86, en el caso del emulador de PocketPC 2002. Por lo tanto, se recomienda comprobar regularmente la aplicación en el verdadero dispositivo de destino, ya que muchos problemas como la alineación de bytes sólo surgen en este dispositivo.

Nota: los binarios del emulador de MQe se proporcionan sólo con fines de desarrollo y no son adecuados para su difusión en un entorno de trabajo real.

Trabajo con hebras

La base de código nativo está diseñada para ser reentrante. La base de código real no utiliza hebras, pero eso no excluye el uso de múltiples hebras en la aplicación. Por ejemplo, se puede crear una hebra de aplicación para llamar repetidamente a `mqeQueueManager_triggerTransmission()`. Si desea utilizar múltiples hebras, no es necesario llamar a ninguna API en concreto.

A pesar de eso, no se trata de un requisito. Se recomienda disponer de un bloque de excepción por hebra. Si se utiliza un bloque de excepción compartido por las hebras, un bloque de excepción para una hebra que se ejecute correctamente puede sobrescribir un bloque de excepción para una hebra que no se ejecute correctamente.

Nota: Debe invocar `mqeSession_initialize` o `mqeSession_terminate` sólo una vez, para que las hebras utilicen una invocación a la API de MQe. Para garantizar esta operación, llámela en la hebra principal antes de que se creen más hebras de aplicaciones. Por ejemplo, **no** utilice las que se indican a continuación:

```
mqeSession_initialize();

mqeSession_initialize();

mqeSession_terminate();

mqeSession_terminate();
```

Convenios de denominación

El convenio de llamadas para todas las API se ha establecido explícitamente en `_cdecl`. Sin embargo, puede utilizar un convenio de invocación distinto en la aplicación.

Manejadores y elementos

En una aplicación es necesario un mecanismo para acceder a los elementos de MQe, como el gestor de colas, los campos, las series de caracteres, etc. Los manejadores utilizan elementos de MQe. El manejador señala un área de memoria utilizada para almacenar la información específica para esa instancia del elemento. Se retiene información para cada elemento. Por lo tanto, se debe tener cuidado al inicializar el manejador correctamente.

Para utilizar un manejador, deberá inicializarlo. Eso se puede hacer llamando a la nueva función del elemento asociado que se va a utilizar. Por ejemplo, para crear una función MQeString, primero deberá llamar a la función `mqeString_new()` y pasar un puntero al manejador `MQeStringHndl` para esa función. La función `mqeString_new()` asigna memoria para la estructura interna y establece los valores por omisión que necesita la función MQeString. Cuando se ha completado correctamente, la función devuelve el manejador, que ahora se puede utilizar en posteriores llamadas a las funciones MQeString.

Cuando se ha acabado con un elemento, es importante llamar a la función `free()` del elemento con la que está asociada el manejador. Las funciones `free()` liberan todos los recursos de los sistemas que utiliza ese elemento. Si se establece el manejador en `NULL`, se genera una pérdida de memoria en la aplicación y el sistema se puede quedar sin recursos. Para evitar esto, establezca el manejador en `NULL` cuando se haya liberado.

Nota: se recomienda no intentar liberar un manejador más de una vez, puesto que se pueden obtener resultados imprevisibles.

Sólo se deben utilizar manejadores con sus elementos asociados. Asimismo, se deben inicializar y liberar de forma adecuada. Las únicas instancias en las que la aplicación no se encarga de inicializar el manejador es cuando un puntero que señala a un manejador se pasa como parámetro de entrada a una API de MQe. En estos casos, se devuelve un manejador completamente inicializado a la aplicación sin que el usuario haya invocado la función new() correspondiente. Un ejemplo de esto es una función `mqeQueueManager_BrowseMessages()`, que tiene un puntero que indica a `MQeVectorHndl` como si fuera un parámetro de entrada. Sin embargo, en estos casos, la aplicación sigue encargándose de la liberación del manejador.

Funciones de memoria de MQe

MQe proporciona las siguientes funciones para la gestión de memoria:

- `mqeMemory_allocate`
- `mqeMemory_free`
- `mqeMemory_reallocate`

Estas funciones utilizan las mismas rutinas de gestión de memoria que se utilizan en la base de código de MQe. Estas rutinas están disponibles para que los programas de las aplicaciones las utilicen. Una aplicación generalmente puede utilizar su propia elección de gestión de memoria. No obstante, algunas llamadas API, por ejemplo `mqeAdministrator_QueueManager_inquire`, tienen que devolver bloques de memoria que contienen información. En ese caso, la memoria se debe liberar mediante la utilización de la función `mqeMemory_free`.

Una ventaja adicional de la utilización de las funciones de memoria de MQe es que su uso se rastrea junto con el proceso de MQe. Sin embargo, no se debe mezclar nunca las llamadas de las asignaciones de memoria. Por ejemplo, no libere la asignación de memoria con la función `mqeMemory_allocate` mediante la llamada en C en tiempo de ejecución `free()`, ya que la aplicación se puede volver inestable.

MQeString

La clase MQeString contiene series de caracteres del sistema y series de caracteres definidas por el usuario. Se trata de una abstracción de series de caracteres utilizadas en la API en C donde es necesaria una serie de caracteres. La clase MQeString permite crear una serie de caracteres en varios formatos, como matrices con puntos de código Unicode, con cada uno de los puntos de código almacenado en un espacio de memoria de 1, 2 o 4 bytes y UTF-8. La implementación actual de la función MQeString da soporte únicamente a formatos externos.

Nota: Aunque se pasen mediante una MQeString, algunas llamadas API necesitan que la serie real se encuentre con el rango ASCII válido.

Series de constantes

Se proporcionan varias series de constantes. Estas se encuentran definidas en los siguientes archivos de cabecera:

- MQe_Admin_Constants.h
- MQe_Adapter_Constants.h
- MQe_Attribute_Constants.h
- MQe_Connection_Constants.h
- MQe_MQe_Constants.h
- MQe_MQeMessage_Constants.h
- MQe_Queue_Constants.h
- MQe_Registry_Constants.h

Constructor

```
MQERETURN osaMQeString_new(MQeExceptBlock* pExceptBlock,
                    MQEVOID*        pInputBuffer,
                    MQETYPEOFSTRING type,
                    MQeStringHndl * phNewString
                         );
```

Esta función crea un nuevo objeto MQeString a partir de un almacenamiento intermedio que contiene datos de tipo carácter. Los datos pueden tener varios formatos soportados, entre ellos matrices de caracteres de un solo byte terminadas en nulo (es decir, las series char* C normales), matrices de caracteres Unicode de doble byte terminadas en nulo, matrices de caracteres Unicode de cuádruple byte terminadas en nulo y matrices UTF-8 terminadas en nulo. El parámetro de tipo indica a la función cuál es el formato del almacenamiento intermedio de entrada.

Destructor

```
MQERETURN osaMQeString_delete(MQeExceptBlock* pExceptBlock,
                              MQeString_*     pString
                             );
```

Esta función destruye un objeto MQeString creado mediante osaMQeString_new, MQeString_duplicate o MQeString_getMQeSubstring.

Captador

```
MQERETURN osaMQeString_get(MQeExceptBlock*      pExceptBlock,
                      MQEVOID*        pOutputBuffer,
                      MQEINT32*       pBufferLength,
                      MQETYPEOFSTRING requiredType,
                      MQECONST MQeStringHndl   hString
                      );
```

Esta función llena un almacenamiento intermedio de caracteres con el contenido de una MQeString que realiza la conversión siempre que es necesario. Sólo se efectúan conversiones sencillas. No se intenta ninguna conversión de página de códigos. Por ejemplo, si se ha transferido una serie SBCS a la serie, el intento de extraer los datos como DBCS (Unicode) funciona correctamente. No obstante, si se han transferido los datos como DBCS y se intentan extraer como SBCS, la operación sólo funcionará si los datos no tienen ningún valor que no pueda representarse con un solo byte. Cuando se utiliza get() para SBCS, DBCS o QBCS, cada carácter se representa mediante su valor de punto de código Unicode.

```
MQERETURN osaMQeString_getSubstring(MQeExceptBlock* pExceptBlock,
                    MQEVOID*        pOutputBuffer,
                    MQEINT32*       pBufferLength,
                    MQETYPEOFSTRING requiredType,
                    MQECONST MQeStringHndl hString,
```

```
                    MQEINT32 from,

                    MQEINT32 to

                    );
```

Esta función es muy parecida a osaMQeString_get excepto en que ésta sólo obtiene una subserie de caracteres (de **from** a **to** inclusive).

```
MQERETURN osaMQeString_getMQeSubstring(MQeExceptBlock* pExceptBlock,

                    MQeStringHndl *      phOutput,

                    MQECONST MQeStringHndl   hString,

                    MQEINT32 from,

                    MQEINT32 to

                    );
```

Esta función es muy parecida a osaMQeString_getSubstring con la diferencia de que devuelve el resultado como una MQeString.

```
MQERETURN osaMQeString_duplicate(MQeExceptBlock  * pExceptBlock,

                    MQeStringHndl * phNewString,

                    MQECONST MQeStringHndl   hString

                    );
```

Esta función duplica una MQeString.

```
MQERETURN osaMQeString_codePointSize(MQeExceptBlock* pExceptBlock,

                    MQEINT32 * pSize,

                    MQECONST MQeStringHndl   hString

                    );
```

Esta función localiza el tamaño de memoria (en bytes) necesario para el carácter más grande de la serie.

```
MQERETURN osaMQeString_getCharLocation( MQeExceptBlock* pExceptBlock,

                                MQEINT32*       pOutIndex,

                    MQECONST MQeStringHndl   hString,

                                MQECHAR32       charToFind,

                                MQEINT32        startFrom,

                    MQEBOOL         searchForward

                    );
```

Esta función devuelve el índice de ubicación (empezando por 0) de la primera aparición de un determinado carácter, especificado como su valor de punto de código Unicode. Puede especificar el punto de inicio de la búsqueda y el sentido de la búsqueda.

Comprobador

```
MQERETURN osaMQeString_isAsciiOnly(MQeExceptBlock*      pExceptBlock,

                        MQEBOOL*             pIsAsciiOnly,

                        MQECONST MQeString_* pString

                        );
```

Esta función determina si la serie contiene algún carácter ASCII que no sea invariable.

```
MQERETURN osaMQeString_equalTo(MQeExceptBlock*      pExceptBlock,

                        MQEBOOL*             pIsEqual,

                        MQECONST MQeString_* pString,

                        MQECONST MQeString_* pEqualToString

                        );
```

Esta función determina si dos series son equivalentes.

```
MQERETURN osaMQeString_isNull(MQeExceptBlock  * pExceptBlock,

                        MQEBOOL * pIsNull,

                        MQECONST MQeStringHndl   hString

                        );
```

Esta función determina si una serie de caracteres es una serie nula. Un manejador NULL también está considerado como una serie nula.

El juego de caracteres de un solo byte (SBCS) es el modo estándar de trabajar con C en una página de código ASCII. Java funciona sólo en Unicode y es posible que haya plataformas a las que tenga que dar soporte que no carguen una página de códigos SBCS, por ejemplo los idiomas de algunos países se representan en DBCS. Puesto que no incluye el puntero de caracteres, el elemento de serie de caracteres permite crear series en una máquina ASCII sin tener que considerar los requisitos Unicode. MQe lleva a cabo cualquier conversión necesaria. Utilice la representación de serie de caracteres UTF-8, puesto que puede hacer frente a cualquier representación de caracteres y realiza la conversión automáticamente. Una vez creada una función MQeString ya no se puede alterar. Sin embargo, existen unas funciones que facilitan la utilización del tipo MQeString. Asimismo, puede crear MQeStrings constantes de forma similar a como se utiliza `#define NAME "mystring"`. La utilización de MQeString garantiza la portabilidad de la aplicación.

Guía: creación de una aplicación básica

Este tema contiene instrucciones paso a paso para crear una aplicación de MQe sencilla. En él se describen los pasos necesarios para crear y configurar el primer gestor de colas y, a continuación, verificar que pueda enviar y recibir mensajes desde otro gestor de colas.

Además de describir lo que es necesario hacer, también indica los mandatos de MQe_Script que se pueden utilizar para realizar cada tarea fácilmente. MQe_Script utiliza los valores por omisión de muchos atributos, que, de otro modo, tendrían que especificarse si se escribiera código equivalente.

MQe_Script está disponible como parte del SupportPac de soporte de servidor en el sitio web de IBM. Este SupportPac incluye documentación exhaustiva sobre el uso de todos los mandatos de MQe_Script, incluyendo información detallada sobre los valores por omisión y explicaciones sobre cómo cambiarlos, si es necesario.

También puede seguir muchos de los pasos que implica este proceso con MQe_Explorer, que se incluye en el mismo SupportPac.

Finalmente, la guía proporciona enlaces con fragmentos de código de ejemplo que muestran cómo llevar a cabo muchos de los pasos mediante programación.

Después de crear e iniciar un gestor de colas, toda la configuración (incluida la creación de colas, las definiciones de conexiones, las definiciones de colas remotas y los escuchas) se realiza mediante mensajes de administración.

1. Crear un gestor de colas (QM1)

Cuando cree un gestor de colas, debe definir los atributos siguientes:

- El nombre del gestor de colas
- El registro público o privado
- La ubicación del registro
- El adaptador del almacén de mensajes
- Las colas por omisión
  - AdminQ
  - AdminReplyQ
  - DeadLetterQ
  - System.default.local.Q

También puede establecer otros atributos (opcionales) en ese momento, como una descripción, un tiempo de espera de canal, un nombre de norma de atributo de canal y una norma de gestor de colas, pero éstos no se incluyen en esta guía.

Creación de QM1 mediante MQe_Script:

Puede utilizar el mandato MQe_Script siguiente para crear un gestor de colas denominado QM1:

```
mqe_script_qm -create -qmname QM1
```

Con este mandato se crea un gestor de colas denominado QM1, con las características siguientes:

- Registro público.
- Una ubicación base C:\Archivos de programa\mqe\java\mqe_script. El registro por omisión y los directorios de colas se encuentran en subdirectorios de esta vía de acceso.
- Utiliza el almacén de mensajes por omisión y guarda la información en el disco.
- Contiene 4 colas por omisión.

También se crea un archivo ini para que la información del gestor de colas se guarde y se pueda iniciar de nuevo pasando la ubicación de este archivo a un método adecuado.

2. Inicio del gestor de colas (QM1)

Después de crear el gestor de colas QM1, tiene que iniciarlo.

Inicio de QM1 mediante MQe_Script:

Puede utilizar el mandato de MQe_Script siguiente para iniciar el gestor de colas QM1:

```
mqe_script_qm -load
```

Si no se proporciona ningún nombre, este mandato inicia el gestor de colas que se acaba de crear. Si no sabe cómo cargar un gestor de colas y especificar el archivo INI, consulte la documentación que se proporciona con MQe_Script.

3. Crear una cola local (Q1)

Cuando haya iniciado el gestor de colas QM1, podrá crear una cola local denominada Q1:

Creación de Q1 mediante MQe_Script:

Puede utilizar el mandato de MQe_Script siguiente para crear una cola local denominada Q1:

```
mqe_script_appq -create -qname Q1
```

Este mandato crea una cola local básica (que se conoce también como *cola de aplicación*) denominada Q1 en el gestor de colas QM1.

4. Crear una definición de conexión

Cuando haya creado la cola local (Q1), tendrá que crear una definición de cola especificando lo siguiente:

- el nombre del gestor de colas con el que desee conectarse (el gestor de colas remoto),
- el puerto en el que estará a la escucha el gestor de colas remoto,
- y el adaptador de comunicaciones.

Creación de una definición de conexión mediante MQe_Script:

Puede utilizar el mandato de MQe_Script siguiente para crear una definición de conexión:

```
mqe_script_condef -create -cdname QM2 -port 1881
```

Este mandato crea una definición de conexión con un gestor de colas denominado QM2, que está a la escucha en el puerto 1881. No es necesario que QM2 exista cuando se cree la conexión, pero debe existir cuando se intente enviar un mensaje a una cola remota en ese gestor de colas. Como no se ha especificado ningún adaptador, el adaptador de Http se utiliza por omisión.

5. Crear una definición de cola remota

Cuando haya creado una definición de conexión, tendrá que crear una definición remota de una cola local en el gestor de colas QM2.

Creación de una definición de cola remota mediante MQe_Script:

Puede utilizar el mandato de MQe_Script siguiente para crear una definición de cola remota:

```
mqe_script_sproxyq -create -qname Q2 -destination QM2
```

Este mandato crea una cola de proxy síncrona, que es una definición remota de una cola local en QM2. No es necesario que QM2 exista cuando se cree la definición de cola remota. No obstante, debe crear una definición de conexión (consulte “4. Crear una definición de conexión”) para poder crear esta definición de cola remota.

6. Crear un escucha (L1)

Cuando haya creado una definición de cola remota, tendrá que crear un escucha.

Creación de un escucha mediante MQe_Script:

Puede utilizar el mandato de MQe_Script siguiente para crear un escucha denominado L1 (en el gestor de colas QM1):

```
mqe_script_listen -create -listenname L1 -port 1882
```

Este mandato crea un escucha para el gestor de colas QM1 que está a la escucha en el puerto 1882. Se utiliza el adaptador de comunicaciones por omisión, que es el adaptador de Http.

7. Iniciar el escucha (L1)

Cuando haya creado un escucha, tendrá que iniciarlo.

Iniciar un escucha mediante MQe_Script:

Puede utilizar el mandato de MQe_Script siguiente para iniciar el escucha L1:

```
mqe_script_listen -start -listenname L1
```

8. Crear un segundo gestor de colas (QM2)

Cuando haya acabado de configurar QM1 (como se muestra en los pasos anteriores de esta guía), tendrá que crear un segundo gestor de colas denominado QM2:

Creación de QM2 mediante MQe_Script:

Puede utilizar el mandato de MQe_Script siguiente para crear un gestor de colas denominado QM2:

```
mqe_script_qm -create -qmname QM2
```

Este mandato crea un gestor de colas denominado QM2, con las características siguientes:

- Registro público.
- Una ubicación base C:\Archivos de programa\mqe\java\mqe_script. El registro por omisión y los directorios de colas se encuentran en subdirectorios de esta vía de acceso.
- Utiliza el almacén de mensajes por omisión y guarda la información en el disco.
- Contiene 4 colas por omisión.

También se crea un archivo ini para que la información del gestor de colas se guarde y se pueda iniciar de nuevo pasando la ubicación de este archivo a un método adecuado.

9. Iniciar QM2

Después de crear el gestor de colas QM2, tiene que iniciarlo.

Inicio de QM2 mediante MQe_Script:

Puede utilizar el mandato de MQe_Script siguiente para iniciar el gestor de colas QM2:

```
mqe_script_qm -load
```

Si no se proporciona ningún nombre, este mandato inicia el gestor de colas que se acaba de crear. Si no sabe cómo cargar un gestor de colas y especificar el archivo INI, consulte la documentación que se proporciona con MQe_Script.

10. Crear una cola local (en QM2) denominada Q2

Cuando haya iniciado el gestor de colas QM2, podrá crear una cola local denominada Q2.

Creación de Q2 mediante MQe_Script:

Puede utilizar el mandato de MQe_Script siguiente para crear una cola local denominada Q2:

```
mqe_script_appq -create -qname Q2
```

Este mandato crea una cola local básica denominada Q2 en el gestor de colas QM2.

11. Crear una definición de conexión (en QM2)

Cuando haya creado la cola local (Q2), tendrá que crear una definición de cola especificando lo siguiente:

- el nombre del gestor de colas con el que desee conectarse (el gestor de colas remoto),
- el puerto en el que estará a la escucha el gestor de colas remoto,
- y el adaptador de comunicaciones.

Creación de una definición de conexión mediante MQe_Script:

Puede utilizar el mandato de MQe_Script siguiente para crear una definición de conexión:

```
mqe_script_condef -create -cdname QM1 -port 1882
```

Este mandato crea una definición de conexión con un gestor de colas denominado QM1, que está a la escucha en el puerto 1882. No es necesario que QM1 exista cuando se cree la conexión, pero debe existir cuando se intente enviar un mensaje a una cola remota en ese gestor de colas. Como no se ha especificado ningún adaptador, el adaptador de Http se utiliza por omisión.

12. Crear una definición de cola remota (en QM2)

Cuando haya creado una definición de conexión, tendrá que crear una definición remota de una cola local en el gestor de colas QM1.

Creación de una definición de cola remota mediante MQe_Script:

Puede utilizar el mandato de MQe_Script siguiente para crear una definición de cola remota:

```
mqe_script_sproxyq -create -qname Q1 -destination QM1
```

Este mandato crea una cola de proxy síncrona, que es una definición remota de una cola local en QM1. No es necesario que QM1 exista cuando se cree la definición de cola remota, pero debe existir antes de transferir a ella un mensaje.

13. Crear un escucha (en QM2) denominado L2

Cuando haya creado una definición de cola remota, tendrá que crear un escucha.

Creación de un escucha mediante MQe_Script:

Puede utilizar el mandato de MQe_Script siguiente para crear un escucha denominado L2 (en el gestor de colas QM2):

```
mqe_script_listen -create -listenname L2 -port 1881
```

Este mandato crea un escucha para el gestor de colas QM2 que está a la escucha en el puerto 1881. Se utiliza el adaptador de comunicaciones por omisión, que es el adaptador de Http.

14. Iniciar el escucha L2 (en QM2)

Cuando haya creado un escucha, tendrá que iniciarlo.

Iniciar un escucha mediante MQe_Script:

Puede utilizar el mandato de MQe_Script siguiente para iniciar el escucha L2:

```
mqe_script_listen -start -listenname L2
```

15. Enviar (PUT) un mensaje de QM1 a QM2

Ahora que ha creado e iniciado los dos gestores de colas, que ha creado las colas y las definiciones de conexión y que ha creado e iniciado los escuchas, ya puede enviar mensajes entre los dos gestores de colas.

Enviar un mensaje mediante MQe_Script:

En QM1, puede utilizar el mandato de MQe_Script siguiente para enviar un mensaje de QM1 a QM2:

```
mqe_script_msg -put -qname Q2 -qmname QM2
```

Este mandato transfiere un mensaje a la cola Q2 del gestor de colas QM2.

16. Recibir (GET) el mensaje en QM2

Ahora que se ha transferido un mensaje a la cola desde QM1, puede recuperarlo desde QM2.

Recibir un mensaje mediante MQe_Script:

Puede utilizar el mandato de MQe_Script siguiente para recuperar el mensaje de la cola:

```
mqe_script_msg -get -qname Q2 -qmname QM2
```

17. Visualización de los detalles de los objetos de MQe

Puede visualizar los detalles de los objetos de MQe que haya creado ejecutando el mandato de MQe_Script `inquireall`. Por ejemplo, para ver información acerca del gestor de colas local, utilice el mandato siguiente:

```
mqe_script_qm -inquireall
```

Con ello se visualizará toda la información acerca del gestor de colas local y se mostrarán los valores por omisión que MQe_Script haya utilizado.

También podrá visualizar información acerca de otros objetos especificando el nombre del objeto. Por ejemplo:

```
mqe_script_condef -inquireall -cdname QM2
```

Ejemplo de una aplicación MQe (HelloWorld)

En este tema se describe cómo crear una aplicación básica (denominada HelloWorld) con las API de MQe en Java y en C. Contiene información sobre el diseño, el desarrollo, la difusión y la ejecución de la aplicación.

Java ″HelloWorld″

En este apartado se describe cómo diseñar, desarrollar, difundir y ejecutar una aplicación ″HelloWorld″ básica en Java.

Diseño de la aplicación en Java

Esta aplicación está destinada a la creación y utilización de un único gestor de colas con una cola local. Esto significa que el mensaje se coloca en la cola local y que después se elimina de ésta.

Se pueden crear gestores de colas para que los utilice un único programa. Cuando este programa se completa, se puede ejecutar un segundo programa que vuelve a utilizar la anterior configuración del gestor de colas.

Normalmente, la configuración de nuevas entidades es un proceso independiente de su uso real. Una vez configuradas, para administrar estas entidades es necesario efectuar un proceso distinto al de su utilización. Este apartado se centra en la utilización más que en la administración.

Si se presupone que la entidad del gestor de colas ya se ha configurado, la aplicación HelloWorld tiene el siguiente flujo de comunicaciones para las bases de código en C y en Java:

1. **Iniciar del gestor de colas**: inicia el gestor de colas según la información que ya se ha creado.
2. **Crear un mensaje**: crea una estructura que se puede utilizar para enviar un mensaje de un gestor de colas a otro.
3. **Transferir a una cola local**: transfiere el mensaje a la cola local.
4. **Obtener de una cola local**: recupera el mensaje de la cola local y comprueba que sea válido.
5. **Concluir**: borra y detiene el gestor de colas.

Desarrollo de la aplicación en Java

El siguiente código figura en la clase `examples.helloworld.Run` en su estado completo. Las soluciones en las que se utilizan las clases de MQe se dividen con frecuencia en varias tareas independientes:

- Instalación de la solución
- Configuración del gestor de colas, dejando la información sobre la configuración en el disco duro local
- Utilización del gestor de colas
- Eliminación del gestor de colas
- Desinstalación de la solución

Antes de leer la información de este capítulo, es necesario configurar un gestor de colas. El programa `examples.helloworld.Configure` muestra la configuración del gestor de colas. El programa `examples.helloworld.Unconfigure` muestra la eliminación del gestor de colas. En este apartado de la documentación se describe cómo se debe utilizar el gestor de colas.

Visión general de examples.helloworld.run:

El método principal controla el flujo de la aplicación hello world. En este código se puede ver que el gestor de colas se ha iniciado, un mensaje se ha colocado en la cola, se ha obtenido un mensaje de una cola y el gestor de colas se ha detenido.

La información de rastreo se puede redirigir a la corriente de datos de salida estándar si en la constante simbólica MQE_TRACE_ON se ha cambiado el valor a 'true'.

```
public static void main(String[ args) {
       try {
             Run me = new Run();

           if (MQE_TRACE_ON) {
             me.traceOn();
           }
           me.start();
           me.put();
           me.get();
           me.stop();
             if (MQE_TRACE_ON) {
               me.traceOff();
           }
        } catch (Exception error) {
              System.err.println("Error: " + error.toString());
           error.printStackTrace();
       }
   }
```

Inicio del gestor de colas:

El programa examples.helloworld.Configure creará una imagen del gestor de colas HelloWorldQM en el disco.

Antes de que se pueda utilizar el gestor de colas, se debe crear una instancia en la memoria y se debe iniciar. El método de inicio en el programa de ejemplo lo hace.

```
public void start() throws Exception {

       System.out.println("Starting the queue manager.");

       String queueManagerName = "HelloWorldQM";
        String baseDirectoryName =
                "./QueueManagers/" + queueManagerName;

       // Create all the configuration
         information needed to construct the
        // queue manager in memory.
          MQeFields config = new MQeFields();

            // Construct the queue manager section parameters.
        MQeFields queueManagerSection = new MQeFields();

        queueManagerSection.putAscii(MQeQueueManager.Name,
             queueManagerName);
        config.putFields(MQeQueueManager.QueueManager,
             queueManagerSection);

               // Construct the registry section parameters.
            // In this examples, we use a public registry.
       MQeFields registrySection = new MQeFields();

             registrySection.putAscii(MQeRegistry.Adapter,
         "com.ibm.mqe.adapters.MQeDiskFieldsAdapter");
        registrySection.putAscii(MQeRegistry.DirName,
                       baseDirectoryName + "/Registry");

        config.putFields("Registry", registrySection);

        System.out.println("Starting the queue manager");
```

```
        myQueueManager = new MQeQueueManager();
       myQueueManager.activate(config);
          System.out.println("Queue manager started.");
    }
```

Para iniciar el gestor de colas, debe conocer como mínimo su nombre, ubicación y el adaptador que se debe utilizar para leer del registro la información sobre configuración del gestor de colas.

La activación del gestor de colas hace que los datos de configuración del disco se lean utilizando el adaptador de campos del disco y, a continuación, el gestor de colas se inicia y se ejecuta, estando así disponible para su utilización.

Creación de un mensaje y colocación en una cola local:

El siguiente código sirve para generar un mensaje, añadir un campo Unicode con un valor de ″Hello World!″ y, a continuación, colocar el mensaje en la cola `SYSTEM.DEFAULT.LOCAL.QUEUE` ubicada en el gestor de colas local `HelloWorldQM`.

```
public void put() throws Exception {
         System.out.println("Putting the test message");
         MQeMsgObject msg = new MQeMsgObject();

       // Add my hello world text to the message.
          msg.putUnicode("myFieldName" , "Hello World!");

         myQueueManager.putMessage(queueManagerName,
        MQe.System_Default_Queue_Name, msg, null, 0L);
         System.out.println("Put the test message");
    }
```

Obtención de una cola local: El siguiente código permite obtener el mensaje ″superior″ de la cola local, `SYSTEM.DEFAULT.LOCAL.QUEUE`, comprueba que se ha obtenido un mensaje con el campo `myFieldName` y muestra el texto retenido en el campo Unicode.

```
public void get() throws Exception {
        System.out.println("Getting the test message.");
        MQeMsgObject msg = myQueueManager.getMessage( queueManagerName,
                                                      MQe.System_Default_Queue_Name,
                                                      null, null, 0L );

        if (msg != null) {
            System.out.println("Got the test message.");

            if (msg.contains("myFieldName")) {
                String textGot = msg.getUnicode("myFieldName");

                System.out.println("Message contained the text '" + textGot + "'");
            }
        }
    }
```

Detención y supresión del gestor de colas:

En este apartado se describe cómo detener un gestor de colas y suprimir su definición.

Detención del gestor de colas

El gestor de colas se puede detener mediante una conclusión controlada.

```
public void stop() throws Exception {
       System.out.println("Stopping the queue manager.");
      myQueueManager.closeQuiesce(QUIESCE_TIME);
      myQueueManager = null;
      System.out.println("Queue manager stopped.");
  }
```

Supresión de la definición del gestor de colas del disco

El programa `examples.helloworld.Unconfigure` se puede utilizar para eliminar el gestor de colas del disco.

Ejecución de la aplicación de Java

En un indicador de mandatos, establezca la variable CLASSPATH de manera que haga referencia a los archivos de clases de MQe. Están disponibles en el directorio Java en el que ha instalado el producto MQe.

Asegúrese de que el shell tiene la capacidad de crear y modificar el directorio `./QueueManagers` en el sistema. Si no es así, cambie el código fuente de los programas `examples.helloworld`, de modo que hagan referencia a un directorio accesible y vuelva a compilar el código java.

Invoque el programa de configuración para crear el gestor de colas. La sintaxis depende de la JVM (Java Virtual Machine) que utilice. La IBM JVM se invoca con el mandato "java", por ejemplo `java examples.helloworld.Configure`. Este mandato crea el gestor de colas en el disco.

Ejecute el programa hellow world `java examples.helloworld.Run`. Esta acción enviará un mensaje a la cola local, obtendrá el mensaje de vuelta y mostrará parte de él.

Ahora se puede destruir el gestor de colas en el disco utilizando `java examples.helloworld.Unconfigure`.

C "HelloWorld"

En este apartado se describe cómo diseñar, desarrollar, difundir y ejecutar una aplicación "HelloWorld" en C.

Diseño de la aplicación en C

Esta aplicación está destinada a la creación y utilización de un único gestor de colas con una cola local. Esto significa que el mensaje se coloca en la cola local y que después se elimina de ésta.

Se pueden crear gestores de colas para que los utilice un único programa. Cuando este programa se completa, se puede ejecutar un segundo programa que vuelve a utilizar la anterior configuración del gestor de colas.

Normalmente, la configuración de nuevas entidades es un proceso independiente de su uso real. Una vez configuradas, para administrar estas entidades es necesario efectuar un proceso distinto al de su utilización. Este apartado se centra en la utilización más que en la administración.

Si se presupone que la entidad del gestor de colas ya se ha configurado, la aplicación HelloWorld tiene el siguiente flujo de comunicaciones para las bases de código en C y en Java:

1. **Iniciar del gestor de colas**: inicia el gestor de colas según la información que ya se ha creado.
2. **Crear un mensaje**: crea una estructura que se puede utilizar para enviar un mensaje de un gestor de colas a otro.
3. **Transferir a una cola local**: transfiere el mensaje a la cola local.
4. **Obtener de una cola local**: recupera el mensaje de la cola local y comprueba que sea válido.

5. **Concluir**: borra y detiene el gestor de colas.

Nota: La base de código en C no dispone de un equivalente de la función Java Garbage Collection. Por lo tanto, el borrado del gestor de colas se destaca mucho más en C.

Desarrollo de la aplicación en C

En este apartado se describe la codificación de alto nivel obligatoria para la aplicación ″HelloWorld″ en C.

Encontrará el código de los ejemplos siguientes en su totalidad en el ejemplo HelloWorld_Runtime.c. El ejemplo contiene código para manejar los datos concretos de ejecución de un programa en un PocketPC, que principalmente incluyen la grabación en un archivo para hacer frente a la falta de opciones de línea de mandatos. Utilice la función de visualización para grabar en un archivo, tal como se muestra en los ejemplos de los siguientes apartados.

Visión general de HelloWorld_Runtime.c:

Para acceder a las API sólo debe incluir un archivo de cabecera. Se debe incluir la definición NATIVE para indicar que no se trata de los enlaces C. Asimismo, se debe definir la plataforma MQE_PLATFORM sobre la que se pretenda ejecutar la aplicación.

```
#define   NATIVE
#define MQE_PLATFORM = PLATFORM_WINCE
#include<published/MQe_API.h>
```

Todo el código, incluidas las declaraciones de variables, se encuentra en el método principal. Para efectuar la comprobación de errores es necesario disponer de estructuras. La estructura MQeExceptBlock se pasa a todas las funciones para que se devuelva la información de error. Además, todas las funciones devuelven un código en el que se indica si se ha producido una anomalía o si el proceso es correcto y que se almacena en la antememoria en una variable local:

```
 /* ... Local return flag */
MQERETURN rc;
 MQeExceptBlock exceptBlock;
```

Se debe crear una cantidad de series de caracteres, por ejemplo para el nombre del gestor de colas:

```
 MQeStringHndl     hLocalQMName;

      ...

if ( MQERETURN_OK == rc ) {
   rc = mqeString_newUtf8(&exceptBlock,
           &hLocalQMName,
                       "LocalQM");
}
```

La primera llamada API realizada es la inicialización de sesión:

```
/* ... Initialize the session */
  rc = mqeSession_initialize(&exceptBlock);
```

Inicio del gestor de colas:

Este proceso consta de dos pasos:

1. Creación del elemento del gestor de colas.
2. Inicio del gestor de colas.

La creación del gestor de colas precisa dos conjuntos de parámetros, uno para el gestor de colas y otro para el registro. Ambos conjuntos de parámetros se inicializan. Es necesario que haya directorios para el *almacén de colas* y el registro.

Nota: Todas las llamadas necesitan un puntero a ExceptBlock y un puntero al manejador del gestor de colas.

```
     if (MQERETURN_OK == rc) {

    MQeQueueManagerParms qmParams  = QMGR_INIT_VAL;
      MQeRegistryParms     regParams = REGISTRY_INIT_VAL;
      qmParams.hQueueStore           = hQueueStore;
    qmParams.opFlags               = QMGR_Q_STORE_OP;

      /* ... create the registry parameters -
        minimum that are required */
      regParams.hBaseLocationName     =  hRegistryDir;
      display("Loading Queue Manager from registry \n");
    rc = mqeQueueManager_new(  &exceptBlock,
                             &hQueueManager,
                              hLocalQMName,
                              &qmParams,
                              &regParams);
}
```

Ahora se puede iniciar el gestor de colas y llevar a cabo operaciones de mensajería:

```
     /* Start the queue manager  */

    if ( MQERETURN_OK == rc ) {
           display("Starting the Queue Manager\n");
       rc = mqeQueueManager_start(hQueueManager,
              &exceptBlock);
    }
```

Creación de un mensaje:

Para crear un mensaje, cree primero un nuevo objeto de campos. En el siguiente ejemplo se añade un único campo. Tenga en cuenta que se pasarán las series de etiquetas de los campos:

```
  MQeFieldsHndl hMsg;

   display("Creating a new message\n");
   rc = mqeFields_new(&exceptBlock,&hMsg);
  if ( MQERETURN_OK == rc ) {
      rc = mqeFields_putInt32(hMsg,&exceptBlk,
          hFieldLabel,42);
  }
```

Colocación de un mensaje en una cola local:

Cuando haya creado el mensaje, podrá colocarlo en una cola local utilizando la función *putMessage*. Tenga en cuenta que se pasarán los nombres del gestor de colas y de las colas. Se pasarán los valores NULL y 0 para la seguridad y para los parámetros de seguridad y de entrega asegurada, puesto que no son necesarios en este ejemplo. Cuando el mensaje se ha colocado, se puede liberar el objeto MQeFields:

```
    if ( MQERETURN_OK == rc ) {
               display("Putting a message \n");
           rc = mqeQueueManager_putMessage(hQueueManager,
                                           &exceptBlock,
                                            hLocalQMName,
                                            hLocalQueueName,
                                           hMsg,
                                            NULL,
                                            0);

            (void) mqeFields_free(hMsg,NULL);
      }
```

Obtención de un mensaje de una cola local:

Cuando el mensaje se ha colocado en una cola, se puede recuperar y comprobar. Existen opciones similares que se pasan a la función getMessage. La diferencia es que se pasa un puntero al manejador de un campo. Se crea un nuevo objeto de campos, eliminando así el mensaje de la cola:

```
MQeFieldsHndl hReturnedMessage;
display("Getting the message back \n");

rc = mqeQueueManager_getMessage(hQueueManager,
                      &exceptBlock,
                      &hReturnedMessage,
                      hLocalQMName,
                      hLocalQueueName,
                            NULL,
                            NULL,
                            0);
}
```

Después de obtener el mensaje, se puede comprobar para saber cuál es el valor que se ha especificado. Para conseguir este valor, utilice la función getInt32. Si el resultado es válido, puede imprimirlo:

```
if (MQERETURN_OK == rc) {
     MQEINT32 answer;
   rc = mqeFields_getInt32(hReturnedMessage,
                            &exceptBlock,
                            &answer,
                            hFieldLabel);

if (MQERETURN_OK == rc) {
     display("Answer is %d\n",answer);
}
else    {
   display("\n\n %s (0x%X) %s (0x%X)\n",
       mapReturnCodeName(EC(&exceptBlock)),
             EC(&exceptBlock),
             mapReasonCodeName(ERC(&exceptBlock)),
             ERC(&exceptBlock)  );
  }

}
```

Conclusión:

Después de eliminar el mensaje de la cola, el gestor de colas se puede detener y liberar. Asimismo, puede liberar las series de caracteres que se han creado. Finalmente, termine la sesión:

```
  (void)mqeQueueManager_stop(hQueueManager,&exceptBlock);
  (void)mqeQueueManager_free(hQueueManager,&exceptBlock);

/* Lets do some clean up */
  (void)mqeString_free(hFieldLabel,&exceptBlock);
  (void)mqeString_free(hLocalQMName,&exceptBlock);
  (void)mqeString_free(hLocalQueueName,&exceptBlock);
  (void)mqeString_free(hQueueStore,&exceptBlock);
  (void)mqeString_free(hRegistryDir,&exceptBlock);


  (void)mqeSession_terminate(&exceptBlock);
```

Compilación:

Para simplificar el proceso de compilación, en el directorio de ejemplos se incluye un archivo makefile. Éste es el archivo makefile exportado desde eMbedded Visual C (EVC). Un archivo de proceso por lotes ejecuta este archivo makefile. Este archivo de proceso por lotes configurará las vías de acceso a los directorios EVC, junto con las vías de acceso a la instalación de MQe. Es posible que tenga que editar el archivo de proceso por lotes, en función de la forma en que desee instalar MQe.

La ejecución del archivo de proceso por lotes compila el ejemplo. Por omisión, el archivo de proceso por lotes se compila para Debug PocketPC 2000 (el procesador ARM o el emulador).

Difusión de la aplicación en C

Para difundir la aplicación "HelloWorld", es necesario crear un gestor de colas. Existen varias formas de hacerlo que se tratan en otros puntos de este centro de información. En este caso, se utiliza el programa HelloWorld_Admin. Ejecútelo tal como se describe a continuación.

Las instrucciones siguientes se aplican tanto al emulador como a un dispositivo real:

1. Copia de todas las DLL en la raíz del dispositivo. Tómelas de los directorios del emulador x86 o ARM.
2. Creación del código de ejemplo mediante el archivo makefile suministrado.

   Nota: Deberá compilar los archivos HelloWorld_Admin.c y HelloWorld_Runtime.c.
3. Copia de estos binarios en el dispositivo o emulador que ejecute PocketPC o el emulador.

Ejecución de la aplicación en C

En este apartado se describe cómo ejecutar la aplicación "HelloWorld" en Java y en C en PocketPC o el emulador.

En este ejemplo se realizan los dos pasos:

1. Creación del gestor de colas. Para efectuar esta tarea, ejecute el programa HelloWorld_Admin. Al ejecutarlo, se creará la representación de disco permanente del gestor de colas.
2. Ejecución del programa HelloWorld_Runtime. Esta tarea inicia un gestor de colas basado en el registro establecido. Para comprobar si el programa ha funcionado correctamente, consulte el archivo de anotaciones que se ha generado. Por omisión, se encuentra en la raíz del dispositivo.

Utilización de las herramientas de desarrollo y administración de MQe

A continuación se detallan algunas de las herramientas que se pueden utilizar para desarrollar o administrar aplicaciones de MQe:

MQe_Explorer
MQe_Explorer proporciona una interfaz gráfica de usuario para la gestión de una red de MQe y su interconexión con MQ. Permite que los gestores de colas de MQe y sus objetos asociados, como colas, conexiones y puentes se configuren de forma local o remota. MQe_Explorer también proporciona una manera fácil de crear gestores de colas locales, que se pueden configurar aún más para cumplir las necesidades de las aplicaciones. También ofrece un entorno de ejecución y depuración para las aplicaciones de MQe. MQe_Explorer está disponible como parte del SupportPac de soporte de servidor.

MQe_Script
MQe_Script es una herramienta basada en línea de mandatos para MQe y es independiente de la plataforma. Permite que los gestores de colas de MQe y sus objetos asociados, como colas, conexiones, escuchas y objetos de puente se configuren de forma local o remota. También se pueden enviar mensajes de prueba a las colas para validar la operación de la red. Como MQe_Explorer, MQe_Script proporciona una manera simple de crear gestores de colas, que, a continuación, se pueden configurar y ampliar para que los utilice la aplicación. MQe_Script está disponible como parte del SupportPac de soporte de servidor.

MQe_Service
MQe_Service es una herramienta basada en asistente para la creación y el funcionamiento de los gestores de colas locales de MQe. Además, permite la configuración automatizada de pasarelas y gestores de colas de MQe, donde se exige que los mensajes pasen entre MQe y redes de MQe.

Plataforma de desarrollo de software Rational

Eclipse

Eclipse es una plataforma abierta con soporte para herramientas de desarrollo de software. Proporciona una infraestructura basada en plug-in que facilita la creación, la integración y la utilización de herramientas de software. Para obtener más información sobre Eclipse, visite:

`http://www.eclipse.org`

Índice