Delphi

MQI Programming Guide

Table of Contents

Introduction

WebSphere MQ offers several interfaces for application programmers to use. Additional interfaces are available as a 'SupportPac' offering. This document describes the SupportPac that delivers MQ functionality to Delphi Pascal programmers as a unit. There are two units in the package, one for 'bindings' mode (MQI) and one for 'client' mode (MQIC).

Implementation

One of the implementations of the WebSphere MQI (MQSeries Interface) on Windows is the 'C' interface. This interface is implemented by a dynamic link library that holds the calls a 'C' programmer can use to interface with MQ. Pascal implementations like Delphi can call external functions/procedures easily by declaring them as 'external' and 'C' versions by adding 'cdecl'. This is how the MQI units define the calls to the 'C' versions.

If that was the only action to take, this supportpac would be a simple one indeed. However, the MQ interface uses a lot of 'C' structures and a huge number of defines. In the MQI Units all the structures are coded as Pascal records and all defines are declared as constants. This facilitates a programming model that looks and feels almost exactly like the 'C' model. This also means that the 'WebSphere MQ Application Programming Reference' can be used as a reference for this SupportPac.

Usage

The SupportPac contains two files called 'MQI.PAS' and 'MQIC.PAS'. These are Pascal sources that should be placed somewhere in the Search Path of Delphi to be included in your Pascal program. The way to include it is like using any unit in Pascal :

uses MQI;

or

uses MQIC;

MQI Calls

The documentation for the MQI calls can be found in the 'WebSphere MQ Application Programming Reference'. The MQI calls in the Pascal unit will resemble the ones described in the MQI 'C' interface, but bear in mind that the 'C' language uses pointers very explicitly and uses another method to determine the length of a string (which is marked by a #0 at the end and therefore 'C' strings are called 'null-terminated'). In Pascal the '@' operator is used to pass a pointer value, so you will see this operator regurarly when calling external procedures that adhere to the 'C' calling interface.

Another thing to notice is the use of special varable types in the MQ interface, which are named in all-capitals like MQCHAR48. These types have been added to the MQI units to remain compatible with the way the 'C' interface is defined.

MQI Record types

The structures you find in the WebSphere MQ Application Programming Reference are defined as records in the MQI units. There is only one record type that differs from the 'C' structure definition, the MQCIH record. The 'C' structure uses a member 'Function' which is a reserved word in Pascal and is abbreviated to 'Func'.

Example

```
MQOD = record
  StrucId              : MQCHAR4;
  Version              : MQLONG;
  ObjectType           : MQLONG;
  ObjectName           : MQCHAR48;
  ObjectQMgrName       : MQCHAR48;
  DynamicQName         : MQCHAR48;
  AlternateUserID      : MQCHAR12;
  RecsPresent          : MQLONG;
  KnownDestCount       : MQLONG;
  UnKnownDestCount     : MQLONG;
  InvalidDestCount     : MQLONG;
  ObjectRecOffset      : MQLONG;
  ResponseRecOffset    : MQLONG;
  ObjectPointer        : MQPTR;
  ResponseRecPointer : MQPTR;
  AlternateSecurityID  : MQBYTE40;
  ResolvedQName        : MQCHAR48;
  ResolvedQMgrName : MQCHAR48;
end;
PMQOD = ^MQOD;
```

MQCONN

The MQCONN call connects an application program to a queue manager. It provides a queue manager connection handle, which is used by the application on subsequent message queuing calls.

Syntax

```
procedure MQCONN ( QmgrName : PMQCHAR48;
                   Hconn : PMQHCONN;
                   Compcode: PMQLONG;
                   Reason : PMQLONG );
```

Example

```
var Hconn : MQHCONN;
    CompCode, Reason : MQLONG;
    QMgrName : MQCHAR48;
begin
    QMgrName := '      ' #0;
    MQCONN ( @QMgrName, @HConn, @Compcode, @Reason);
    if CompCode <> MQCC_OK then
      writeln( 'Connect Failed' );
end;
```

MQCONNX

The MQCONNX call connects an application program to a queue manager. It provides a queue manager connection handle, which is used by the application on subsequent MQ calls.

The MQCONNX call is similar to the MQCONN call, except that MQCONNX allows options to be specified to control the way that the call works.

Syntax

```
procedure MQCONNX ( QMgrName : PMQCHAR48;
                    ConnectOpts : PMQCNO;
                    Hconn : PMQHCONN;
                    Compcode : PMQLONG;
                    Reason : PMQLONG);
```

Example

```
var Hconn : MQHCONN;
    ConnectOpts : MQCNO;
    CompCode, Reason : MQLONG;
    QMgrName : MQCHAR48;
begin
    QMgrName := '      ' #0;
    ConnectOpts := MQCNO_DEFAULT;
    MQCONNX ( @QMgrName, @ConnectOpts, @HConn, @Compcode, @Reason);
    if CompCode <> MQCC_OK then
      writeln( 'Connect Failed' );
end;
```

MQDISC

The MQDISC call breaks the connection between the queue manager and the application program, and is the inverse of the MQCONN or MQCONNX call.

Syntax

```
procedure MQDISC ( Hconn : PMQHCONN;
                   Compcode : PMQLONG;
                   Reason : PMQLONG )
```

Example

```
var Hconn : MQHCONN;
    CompCode, Reason : MQLONG;
begin
  { You should have called MQCONN or MQCONNX previously ofcourse }
  MQDISC ( @Hconn, @Compcode, @Reason );
   if CompCode <> MQCC_OK then
     writeln( 'Disconnect Failed' );
end;
```

MQOPEN

The MQOPEN call establishes access to an object. The following types of object are valid:

- Queue (including distribution lists)
- Namelist
- Process definition
- Queue manager

Syntax

```
procedure MQOPEN ( Hconn : MQHCONN;
                   ObjDesc : PMQOD;
                   Options : MQLONG;
                   Hobj : PMQHOBJ;
                   CompCode : PMQLONG;
                   Reason : PMQLONG);
```

Example

```
var Hconn : MQHCONN;
    Hobj : MQHOBJ;
    ObjDesc : MQOD;
    Options, CompCode, Reason : MQLONG;
 begin
   { Calling the MQCONN procedure to get a valid connection handle was left out ! }
   Options := MQOO_FAIL_IF_QUIESCING + MQOO_OUTPUT;
   ObjDesc := MQOD_DEFAULT;
   with ObjDesc do
   begin
     ObjectName := 'LOCALQ.QUEUE'#0;
   end;
   MQOPEN (Hconn, @ObjDesc, Options, @HObj, @Compcode, @Reason );
   if CompCode <> MQCC_OK then
     writeln( 'Open Failed' );
end;
```

MQCLOSE

The MQCLOSE call relinquishes access to an object, and is the inverse of the MQOPEN call.

Syntax

```
procedure MQCLOSE ( Hconn : MQHCONN;
                    Hobj : PMQHOBJ;
                    Options : MQLONG;
                    Compcode, Reason : PMQLONG);
```

Example

```
var Hconn : MQHCONN;
    Hobj : MQHOBJ;
    Options, CompCode, Reason : MQLONG;
begin
   { Calling the MQOPEN procedure to get a valid object handle was left out ! }
   Options := MQCO_NONE;
   MQCLOSE ( Hconn, @Hobj, Options, @Compcode, @Reason);
end;
```

MQPUT1

The MQPUT1 call puts one message on a queue. The queue need not be open.

Syntax

```
procedure MQPUT1 ( Hconn : MQHCONN;
                   ObjDesc : PMQOD;
                   MsgDesc : PMQMD;
                   PutMsgOptions : PMQPMO;
                   BufferLength : MQLONG;
                   Buffer : PChar;
                   CompCode : PMQLONG;
                   Reason : PMQLONG);
```

Example

```
var Hconn : MQHCONN;
    ObjDesc : MQOD;
    MsgDesc : MQMD;
    PutMsgOptions : MQPMO;
    BufferLength : MQLONG;
    Buffer : array [0..1024] of Char;
    CompCode, Reason : MQLONG;
begin
   { Calling the MQCONN procedure to get a valid connection handle was left out ! }
    Buffer := 'This message was sent from Delphi'#0;
    BufferLength := 33;
    MQ_od := MQOD_DEFAULT;
    with MQ_od do
    begin
     ObjectName := 'LOCALQ.QUEUE'#0;
    end;
   MsgDesc := MQMD_DEFAULT;
   PutMsgOptions := MQPMO_DEFAULT;
   MQPUT1 ( Hconn, @ObjDesc, @MsgDesc, @PutMsgOptions,
```

```
                BufferLength, @Buffer, @Compcode, @Reason );
    if CompCode <> MQCC_OK then
      writeln( 'PUT1 Failed' );
end;
```

MQPUT

The MQPUT call puts a message on a queue or distribution list. The queue or distribution list must already be open.

Syntax

procedure MQPUT (Hconn : MQHCONN;
Hobj : MQHOBJ;
MsgDesc : PMQMD;
PutMsgOptions : PMQPMO;
BufferLength : MQLONG;
Buffer : PChar;
CompCode: PMQLONG;
Reason: PMQLONG) **;**

Example

```
var Hconn : MQHCONN;
    Hobj : MQHOBJ;
    MsgDesc : MQMD;
    PutMsgOptions : MQPMO;
    BufferLength : MQLONG;
    Buffer : array [0..1024] of Char;
    CompCode, Reason : MQLONG;
begin
   { Calling the MQCONN or MQCONNX and MQOPEN procedure to get a valid  }
   { connection handle and object handle was left out !                 }

    Buffer := 'This message was sent from Delphi'#0;
    BufferLength := 33;
    MsgDesc := MQMD_DEFAULT;
    PutMsgOptions := MQPMO_DEFAULT;
    MQPUT ( Hconn, Hobj, @MsgDesc, @PutMsgOptions,
                BufferLength, @Buffer, @Compcode, @Reason);
   if CompCode <> MQCC_OK then
     writeln( 'PUT Failed' );
end;
```

MQGET

The MQGET call retrieves a message from a local queue that has been opened using the MQOPEN call.

Syntax

```
procedure MQGET ( Hconn : MQHCONN;
                  Hobj : MQHOBJ;
                  MsgDesc : PMQMD;
                  GetMsgOptions : PMQGMO;
                  BufferLength : MQLONG;
                  Buffer : Pchar;
                  DataLength : PMQLONG;
                  CompCode: PMQLONG;
                  Reason : PMQLONG);
```

Example

```
var Hconn : MQHCONN;
    Hobj : MQHOBJ;
    MsgDesc : MQMD;
    GetMsgOptions : MQGMO;
    BufferLength, DataLength : MQLONG;
    Buffer : array [0..1024] of Char;
    CompCode, Reason : MQLONG;
begin
   { Calling the MQCONN or MQCONNX and MQOPEN procedure to get a valid  }
   { connection handle and object handle was left out !                 }

   BufferLength := 1024; DataLength := 0;
   MsgDesc := MQMD_DEFAULT;
   GetMsgOptions := MQGMO_DEFAULT;
   MQGET ( Hconn, Hobj, @MsgDesc, @GetMsgOptions,
           BufferLength, Buffer, @DataLength, @Compcode, @Reason);
   if CompCode <> MQCC_OK then
     writeln( 'GET Failed' );
end;
```

MQINQ

The MQINQ call returns an array of integers and a set of character strings containing the attributes of an object. The following types of object are valid:

- Queue
- Namelist
- Process definition
- Queue manager

Syntax

```
procedure MQINQ ( Hconn : MQHCONN;
                  Hobj : MQHOBJ;
                  SelectorCount : MQLONG;
                  Selectors : PMQLONG;
                  IntAttrCount : MQLONG;
                  IntAttrs : PMQLONG;
                  CharAttrLength : MQLONG;
                  CharAttrs : PMQCHAR;
                  CompCode : PMQLONG;
                  Reason : PMQLONG);
```

Example

```
var Hconn : MQHCONN;
    Hobj : MQHOBJ;
    ObjDesc : MQOD;
    Selectors : array [0..2] of MQLONG;
    Options, CompCode, Reason : MQLONG;
    QMgrName : MQCHAR48;
begin
   QmgrName := '      '#0;
   MQCONN ( @QMgrName, @HConn, @Compcode, @Reason);
   if CompCode <> MQCC_OK then
     writeln( 'Connect Failed' );

   ObjDesc := MQOD_DEFAULT;

   with ObjDesc do
```

```
    begin
      ObjectQmgrName := QMgrName;
      ObjectType := MQOT_Q_MGR;
    end;

    Options := MQOO_INQUIRE + MQOO_FAIL_IF_QUIESCING;

    MQOPEN (Hconn, @ObjDesc, Options, @HObj, @Compcode, @Reason );
    if CompCode <> MQCC_OK then
      writeln( 'Open Failed' );


    Selectors[0] := MQCA_Q_MGR_NAME;

    MQINQ ( Hconn, Hobj, 1, @Selectors, 0, nil, MQ_Q_MGR_NAME_LENGTH,
              Buffer, @Compcode, @Reason );
    if CompCode <> MQCC_OK then
      writeln( 'Inquire Failed' );
end;
```

MQSET

The MQSET call is used to change the attributes of an object represented by a handle. The object must be a queue.

Syntax

```
procedure MQSET ( Hconn : MQHCONN;
                  Hobj : MQHOBJ;
                  SelectorCount : MQLONG;
                  Selectors : PMQLONG;
                  IntAttrCount : MQLONG;
                  IntAttrs : PMQLONG;
                  CharAttrLength : MQLONG;
                  CharAttrs : PMQCHAR;
                  CompCode : PMQLONG;
                  Reason : PMQLONG);
```

MQBEGIN

The MQBEGIN call begins a unit of work that is coordinated by the queue manager, and that may involve external resource managers.

Syntax

procedure MQBEGIN (Hconn : MQHCONN;
BeginOptions : PMQBO;
CompCode : PMQLONG;
Reason : PMQLONG);

MQCMIT

The MQCMIT call indicates to the queue manager that the application has reached a syncpoint, and that all of the message gets and puts that have occurred since the last syncpoint are to be made permanent. Messages put as part of a unit of work are made available to other applications; messages retrieved as part of a unit of work are deleted.

Syntax

procedure MQCMIT (Hconn : MQHCONN;
CompCode : PMQLONG;
Reason : PMQLONG);

MQBACK

The MQBACK call indicates to the queue manager that all the message gets and puts that have occurred since the last syncpoint are to be backed out. Messages put as part of a unit of work are deleted; messages retrieved as part of a unit of work are reinstated on the queue.

Syntax

procedure MQBACK (Hconn : MQHCONN;
CompCode : PMQLONG;
Reason : PMQLONG);