

Introduction to Microsoft Message Queuing Services (MSMQ)

Distributed applications run on two or more computers. They communicate with one another by passing data over machine boundaries through appropriate network protocols. Most of these protocols use synchronous technologies, such as *Remote Procedure Calls* (RPC) and DCOM. The synchronous process model has a number of limitations, however. Message queuing provides an asynchronous programming model and a loosely coupled environment for different components of distributed applications.

This chapter teaches you the following:

- The limitations of synchronous processing
- Message queuing technology and Microsoft Message Queuing Services (MSMQ)
- MSMQ architecture
- How to write MSMQ applications in Visual Basic

Limitations of the Synchronous Processing Model



EXAMPLE

In a synchronous processing model, components interact with one another in a tightly coupled manner. DCOM applications are examples of synchronous processing, as shown in Figure 3.1.

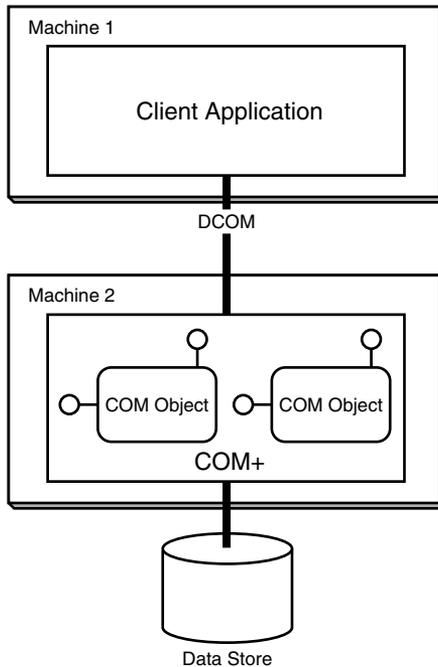


Figure 3.1: DCOM applications use the synchronous processing model.

In Figure 3.1, the client application in machine 1 interacts with the COM objects that are running on machine 2 using DCOM. This synchronous processing model has several limitations:

- A synchronous processing system requires a reliable network connection. As in Figure 3.1, the client has no way to communicate with the server when the network is disconnected.
- In a synchronous processing system, both communicating parties must be available at the same time. In Figure 3.1, if the COM objects on machine 2 are not up and running, calls from the client application fail.

- In Figure 3.1, when the client application makes a call to the COM object on the server, it must wait for the server to finish its processing. If the server takes a long time to process a client call, the client application is blocked (or frozen) until the server finishes processing.
- If machine 2 in Figure 3.1 is shut down for some reason, the calls from the client to the server fail. Therefore, the synchronous processing model is not fault tolerant and thus is not a robust architecture.

Message Queuing and MSMQ

This section introduces message queuing and MSMQ and explains why message queuing technology and products can overcome the shortcomings of the synchronous processing model.

Asynchronous Processing and Message Queuing

As you can see from the previous discussion, a tightly coupled architecture is not suitable for today's distributed applications, such as Windows DNA applications. In a typical DNA application, having a reliable network connection, available servers, and so on is not always feasible. Message queuing, on the other hand, provides an *asynchronous* processing model that addresses the limitations of the synchronous processing model.



EXAMPLE

Message queuing products use a *store-and-forward* mechanism to handle the interaction between different applications. In a typical message queuing system, like the one in Figure 3.2, instead of calling the server directly as in DCOM applications, the client sends data in the form of a message to a temporary data store, which is called a *queue*. The underlying message queuing service internally forwards the message to another queue on the server. A receiver application on the server then picks up the message from the queue and invokes the server to process.

As shown in Figure 3.2, the request of the client is processed in a loosely coupled, asynchronous manner. An asynchronous system such as message queuing can be configured in such a way that if the network is down, the message stays in the queue on the client machine and the data is not lost. The message queuing service forwards the message to the server queue if the network connection becomes available again. The receiver application on the server machine can pick up and process the message at another time. Finally, as long as the client application sends the message to the queue, it is ready to do whatever else it needs to do because it's not blocked by the server process any more.

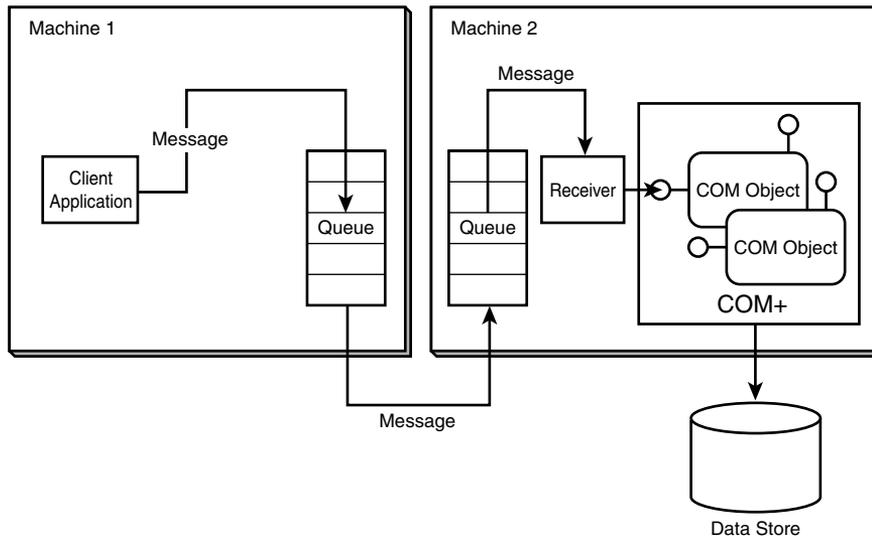


Figure 3.2: A message queuing system.

NOTE

The terms *client* and *client application* here simply mean the message sender. They are relative terms. The roles of the sender and the receiver can be reversed.

TIP

The configuration described here is called an *independent client* in MSMQ. In the IBM MQSeries system, you can use clustering queue managers to achieve the same results. IBM MQSeries, which is IBM's message queuing product, offers capabilities comparable to MSMQ. I will discuss only MSMQ in this book, because this chapter is intended to give you all the background information for Chapter 9, "Queued Components." Queued Components is an important COM+ services that uses MSMQ to achieve messaging queuing functionality.

MSMQ

Message queuing products are sometimes referred to as *Message-Oriented Middleware* (MOM). Microsoft Message Queue Services 2.0 is now an integrated part of Windows 2000 component services; it is the Microsoft implementation of MOM technology.

Applications developed for MSMQ can communicate across heterogeneous networks and with computers that may be offline. MSMQ provides guaranteed message delivery, efficient routing, security, transactional support, and priority-based messaging.

TIP

In Microsoft Windows 2000 documentation, MSMQ 2.0 is referred to as *message queuing*. In Microsoft Platform SDK documentation, both terms (MSMQ, and message queuing) are used.

MSMQ Architecture

Depending on your Windows 2000 configuration, MSMQ can be used in a domain environment or a workgroup environment. The difference is that for MSMQ, a domain environment includes domain controllers that provide a directory service, such as Active Directory, whereas a workgroup environment does not provide such a directory service.

Domain Environment

In a domain environment, an MSMQ network is a group of Windows 2000 *sites*, connected by *routing links*. Sites map the physical structure of a network, whereas domains map the logical structure of an organization. Sites and domain structures are independent of each other. A single site can have multiple domains, whereas a single domain can also have multiple sites. In Windows 2000, a site is defined as a set of computers in one or more IP subnets. Routing links are logic communication links created by MSMQ to route messages between different sites. In MSMQ, a computer that can provide message queuing, routing, and directory services to client computers is called an *MSMQ server*. A routing link is made up of MSMQ servers, one on each site.

CAUTION

Don't confuse routing links with *site links*. Routing links are used by MSMQ to route messages between sites, whereas site links are used by domain controllers to replicate Active Directory between sites.

Workgroup Environment

An MSMQ computer can also run in a workgroup environment that is not part of a domain. There are several restrictions, however. All the benefits provided by Active Directory Services are not available.

First, messages cannot be routed by an MSMQ server; a direct connection with the destination server is required.

Second, you can create and manage only *private queues* on a local computer. You cannot view or manage *public queues*. You can, however, send messages to or read messages from private queues, provided that a direct connection to the destination MSMQ server is specified.

NOTE

In MSMQ 2.0, public queues are those published in Active Directory and can be accessed anywhere in the Active Directory forest. Private queues are not published in Active Directory and can be accessed only by MSMQ applications that know the full pathname or the format name of the queue. Public queues are persistent. Private queues are lightweight and more suitable for offline operations in which the directory services may not be available.

Finally, you cannot use internal certificates to send authenticated messages. Instead, you must use an external certificate.

Queues

In MSMQ, queues are temporary storage locations for different types of messages. Queues can be logically divided into two groups: *application queues* and *system queues*. Application queues are created by applications. System queues are created by MSMQ.

TIP

Application queues can also be created using the Computer Management MMC snap-in.

**EXAMPLE**

Figure 3.3 shows the different types of queues in the Message Queuing services in the Computer Management snap-in.

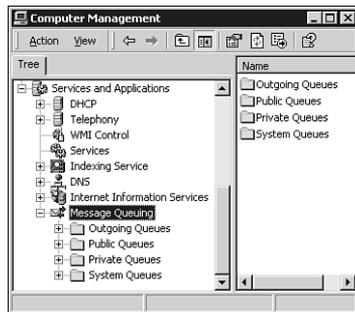


Figure 3.3: Message Queuing services in the Computer Management snap-in.

TIP

If the Message Queuing service is not started yet for some reason, you cannot see it under the Services and Applications node in the Computer Management snap-in. You can manually start the Message Queuing service by using the Component Services snap-in. From Services, locate and right-click the Message Queuing service; then select Start (as shown in Figure 3.4).

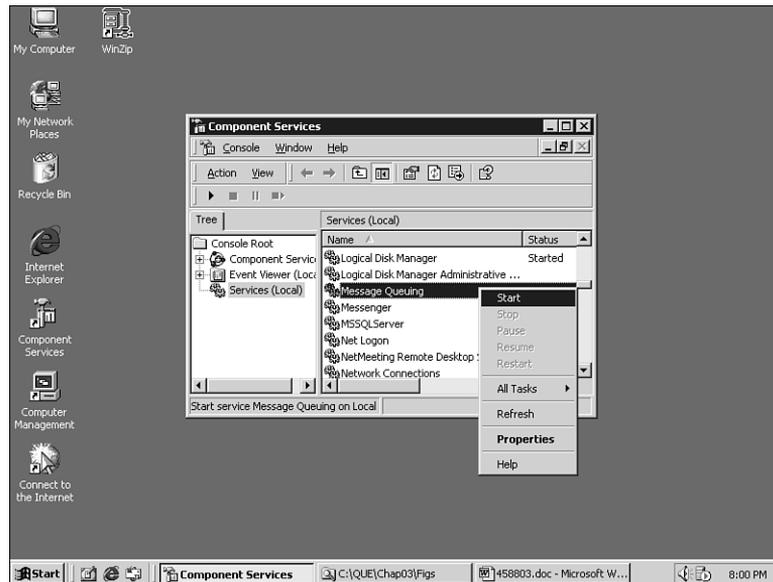


Figure 3.4: Starting the Message Queuing service in the Component Services snap-in.

In Figure 3.3, you may have noticed another type of queue: the *outgoing queue*. Those queues are used for offline operations in which directory service is not available. When MSMQ on a client machine is configured for offline use, it is called an *independent client*. When MSMQ on a client machine is configured for real-time access support, it is called a *dependent client*.

APPLICATION QUEUES

Application queues include *message queues*, *administration queues*, *response queues*, and *report queues*. These queues are created by applications.



EXAMPLE

Message queues allow applications to exchange data through messages. Applications can send messages to and receive them from message queues. Message queues can be either public or private. Figure 3.5 shows an example of a message queue called TestQueue that is created as a private queue.

CAUTION

When you create a queue from an application, it is always displayed in lowercase under Message Queuing in the Computer Management snap-in. However, the names in MSMQ are case sensitive, so be extremely careful in your code when you refer to a queue. For example, if you create a queue called MyQueue, it shows up in MSMQ as myqueue. In your code, however, you still need to access this queue by using MyQueue. You get an error if you refer it as myqueue.

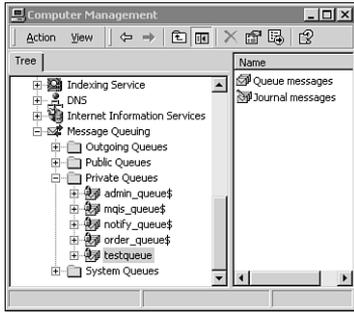


Figure 3.5: A message queue.



Administration queues, which are specified by the sending application, store system-generated acknowledgment messages sent by MSMQ. If you specify the administration queue when you send a message, MSMQ generates an acknowledgment message and sends it to the administration queue specified, indicating whether the original message was successfully sent (see Figure 3.6).

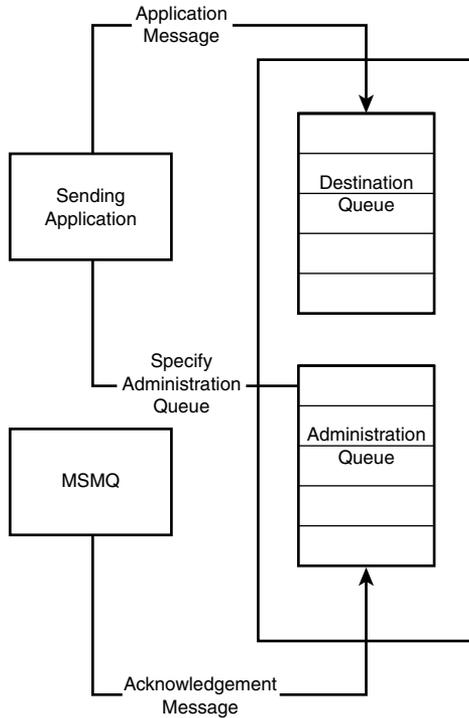


Figure 3.6: An administration queue.



EXAMPLE

Response queues are specified by the sending application and used by the receiving application to send response messages back to the sending application (see Figure 3.7).

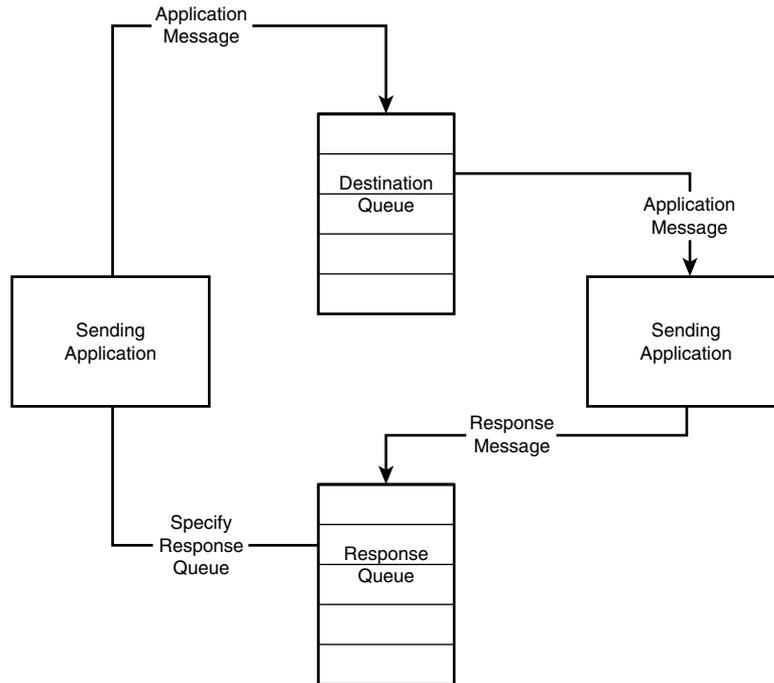


Figure 3.7: A response queue.

Report queues track the progress of messages as they move through the enterprise. When the sending application enables tracking and specifies a report queue, MSMQ sends *report messages* to the report queue. A report message is a system message that is generated each time an application message passes through an MSMQ routing server.

SYSTEM QUEUES



EXAMPLE

System queues are created either by MSMQ or the MSMQ administrator. System queues contain *journal queues* and *dead-letter queues*. Whenever an application queue is created, MSMQ automatically create a journal to track the messages that are removed from the queue. Dead-letter queues store messages that could not be delivered. MSMQ provides two dead-letter queues for each computer: one for nontransactional messages and the other for transactional messages. Figure 3.8 shows system queues.

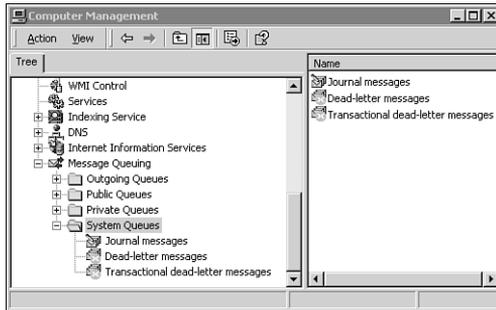


Figure 3.8: System queues.

Messages

MSMQ messages are data exchanged between applications. Messages can be generated by MSMQ applications or by MSMQ itself. This chapter addresses only application-generated messages and some of their important properties.



EXAMPLE

For each message, MSMQ generates and assigns a *message identifier*. The identifier, or ID, of a message is unique on the computer where the message resides and can be used along with other message properties to identify a message. Figure 3.9 shows the property page of a message with its message identifier highlighted.

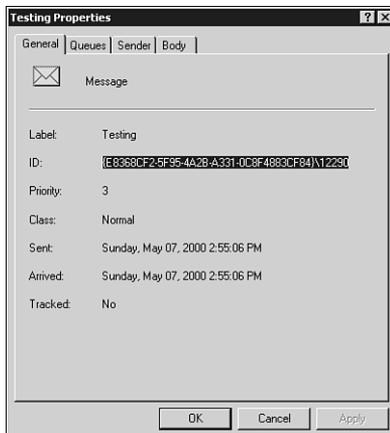


Figure 3.9: A message identifier (ID).

A message identifier is composed of the machine GUID of the computer that sent the message and an identifier that is unique to the computer. For example, in Figure 3.9, the message identifier is

```
{E8368CF2-5F95-4A2B-A331-0C8F4883CF84}\12290
```

The `Label` property of a message is used to describe the message, much like the subject of an email. The `Label` of the message in Figure 3.9 is `Testing`.

Unlike an email message, however, the `Body` property of a message is not limited to string data types. The body of a message is a variant data type. It can be literally any data type, including string, date, numeric, currency, or array of bytes. The body of a message can be a persistent object such as an Excel spreadsheet or even an ADO recordset.

Journaling

MSMQ journaling allows you to keep track of messages. The two types of MSMQ journaling are *source journaling* and *target journaling*. Source journaling tracks messages sent by a computer, whereas target journaling tracks messages removed from a queue.

Programming MSMQ in Visual Basic

MSMQ provides both API functions and a COM interface for developers to interact with it programmatically. This book focuses on the COM interface. I'll first introduce MSMQ COM objects. Then I'll show you some basic MSMQ examples followed by a couple of advanced MSMQ programming examples. Finally, I'll give you an asynchronous ordering example to demonstrate how to use MSMQ in real-world scenarios.

MSMQ COM Object Model

MSMQ provides a set of COM objects that allow applications to access and manage message queuing. The three most important MSMQ COM objects are `MSMQQueueInfo`, `MSMQQueue`, and `MSMQMessage`. Their relationship is illustrated in Figure 3.10.

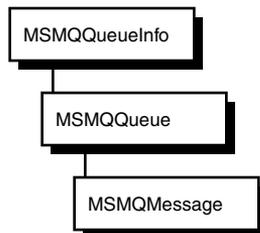


Figure 3.10: Important MSMQ COM objects.

`MSMQQueueInfo`, which provides queue management, allows you to create or delete a queue, open an existing queue, or manipulate the properties of a queue.

MSMQQueue represents an open instance of an MSMQ queue. It provides a cursor-like mechanism for traversing the messages in an open queue. Like a database cursor, at any give moment, it points to a particular message in the queue.

MSMQMessage provides properties to define the behavior of a message and the methods for sending the message to the queue.

Other MSMQ COM objects support additional functionalities:

- The MSMQApplication object provides methods or properties to retrieve information from the MSMQ machine. For example, the IsDSEnabled property tells you whether MSMQ is using the directory service on the computer.
- The MSMQQueueInfos and MSMQQuery objects allow you to get information on public queues. The MSMQQueueInfos object represents a set of MSMQ public queues and allows you to select a specific public queue from a collection of queues. The MSMQQuery object allows you to query the directory service for existing public queues.
- The MSMQEvent object provides an interface for you to implement a single event handler that supports multiple queues.
- The MSMQTransaction, MSMQTransactionDispenser, and MSMQCoordinatedTransactionDispenser objects allow you to manage internal and external MSMQ transactions.

Basic MSMQ Examples



EXAMPLE

To work with MSMQ, you need to set a reference to the Microsoft Message Queue Object Library in a Visual Basic project, as shown in Figure 3.11. Later, in the code samples, you will notice the syntactical difference between creating public and private queues.

The first example, Listing 3.1, creates a queue, opens the queue for send access, and puts a testing message in the queue.

CAUTION

Depending in which directory you put the sample code of this chapter, when you load the source code you may experience an error, “Could Not Create Reference....” If this error occurs, you should reset the references to “Microsoft Message Queue 2.0 Object Library” by select Project, References menu option. This object library is usually located in “\WINNT\system32\MQOA.dll”.

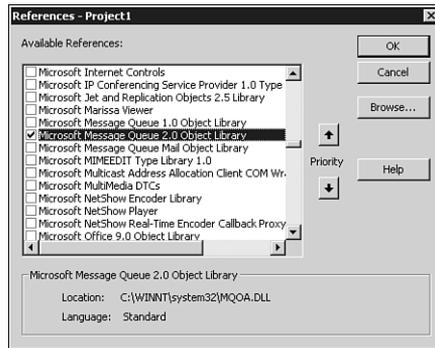


Figure 3.11: Setting a reference to the Microsoft Message Queue Object Library.



Listing 3.1 Creating and Opening a Queue and Sending a Message

```
Public Sub SendQueueMessage()
    '=====
    'In this sub routine, we will create a queue, open
    'the queue and send a testing message to the queue.
    '=====
    'Enable the error handler
    On Error GoTo SendQueueMessage_Err
    'Declare variables for MSMQ objects.
    Dim oQInfo As MSMQ.MSMQQueueInfo
    Dim oQueue As MSMQ.MSMQQueue
    Dim oMessage As MSMQ.MSMQMessage

    'Initialize the MSMQQueueInfo object.
    Set oQInfo = New MSMQQueueInfo
    'we use a conditional compilation constant
    'to take care of both public and private queues.
    #If bUseDS Then
        'If directory service is used, we can create
        'a public queue.
        oQInfo.PathName = ".\TestingQueue"
    #Else
        'Else we can only create a private queue.
        oQInfo.PathName = ".\PRIVATE$\TestQueue"
    #End If
    'Now we are ready to create the queue.
    oQInfo.Label = "Testing Queue"
    oQInfo.Create
    'Open the queue for send access.
    Set oQueue = oQInfo.Open(MQ_SEND_ACCESS, MQ_DENY_NONE)
```

Listing 3.1 continued

```

'If the queue is opened sccessfully, we send a
'testing messge to it.
If oQueue.IsOpen Then
    'Initialize the MSMQMessage object.
    Set oMessage = New MSMQMessage
    'Prepare the message and send to the queue.
    With oMessage
        .Label = "Testing Message"
        .Priority = 5 'Default priority is 3.
        .Body = "Testing Message"
        .Send oQueue
    End With
Else
    'Queue is not open, report the error and get out.
    MsgBox "The queue is not open!"
    Exit Sub
End If
'If everything is ok, close the queue and get out.
oQueue.Close
MsgBox "The message is sent!"
Exit Sub
SendQueueMessage_Err:
'If the queue already exist when we try to create it, '
'ignore the error and move on.
If Err.Number = MQ_ERROR_QUEUE_EXISTS Then
    Resume Next
End If
'Handling other errors.
MsgBox Err.Description
End Sub

```

In Listing 3.1, you use a Visual Basic *conditional compilation constant* that you set on the Make tab of the project's property page (see Figure 2.12). This way, you can have a single code base to handle creating both public and private queues.

The Open method of the MSMQQueueInfo object takes two parameters: Access Mode and Shared Mode. Access Mode can be MQ_SEND_ACCESS, MQ_RECEIVE_ACCESS, or MQ_PEEK_ACCESS. Shared Mode can be MQ_DENY_NOEN (the default) or MQ_DENY_RECEIVE_SHARE. Note that you set the priority to 5 to overwrite the default priority (3). MSMQ puts a message with higher priority in front of a message with lower priority. MSMQ message priorities range from 0 to 7. Also note that in the error handler, you test whether the error was caused by trying to create an already existing queue; then you ignore the error and continue execution of the next line of code. Figure 3.13

shows that the queue is created, and a testing message with a priority of 5 appears in the queue.

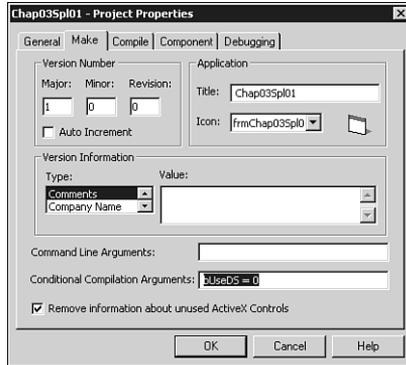


Figure 3.12: Setting the conditional compilation constant.

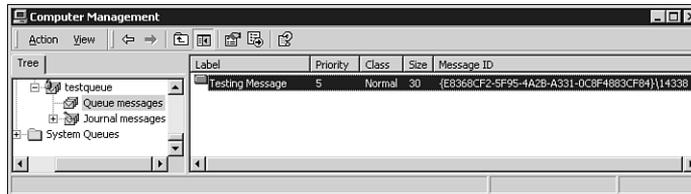


Figure 3.13: A message is sent to the queue.



EXAMPLE

The next example, Listing 3.2, opens an existing queue, retrieves a message from the queue, and prints the contents of the message (label and body) in the debug window.

Listing 3.2 Opening an Existing Queue and Receiving a Message

```
Public Sub ReceiveQueueMessage()
    '=====
    'In this sub routine, we open an existing queue
    'retrieve the message and print to debug window.
    '=====
    'Enable the error handler
    On Error GoTo ReceiveQueueMessage_Err
    'Declare variables for MSMQ objects.
    Dim oQInfo As MSMQ.MSMQQueueInfo
    Dim oQueue As MSMQ.MSMQQueue
    Dim oMessage As MSMQ.MSMQMessage

    'Initialize the MSMQQueueInfo object.
    Set oQInfo = New MSMQQueueInfo
```

Listing 3.2 continued

```

'we use a conditional compilation constant
'to take care of both public and private queues.
#If bUseDS Then
    oQInfo.PathName = ".\TestingQueue"
#Else
    oQInfo.PathName = ".\PRIVATE$\TestQueue"
#End If
'Open the queue for receive access.
Set oQueue = oQInfo.Open(MQ_RECEIVE_ACCESS, MQ_DENY_NONE)
'If the queue is opened successfully,
'we retrieve the message.
If oQueue.IsOpen Then
    'Retrieve the message and print it.
    Set oMessage = oQueue.ReceiveCurrent(ReceiveTimeout:=1000)

    Debug.Print "Message Label: " & oMessage.Label & vbCrLf
    Debug.Print "Message Body: " & oMessage.Body
Else
    'Queue is not open, report the error and get out.
    MsgBox "The queue is not open!"
    Exit Sub
End If
'If everything is ok, we are out of here.
Exit Sub
ReceiveQueueMessage_Err:
    MsgBox Err.Description
End Sub

```

CAUTION

The code in Listing 3.2 will only work if there is a message in the queue. Otherwise you will get an “Object variable or With block variable not set” error message. This is because if there is no message in the queue, the `ReceiveCurrent()` will time out and the next line tries to access the `oMessage` object which is set to `Nothing`.

In Listing 3.2, you use the `Receive` method of the `MSMQQueue` object. Messages are removed from the queue after the `Receive` method is called. This procedure is called *dequeuing*. Note that you use a Visual Basic named argument syntax to specify the timeout value to one minute. Figure 3.14 shows the result.

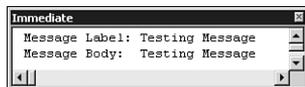


Figure 3.14: A message is received from the queue.

**EXAMPLE**

The following example, Listing 3.3, shows you how to locate a public queue that is registered in Active Directory and delete it if you find one.

Listing 3.3 Locating a Public Queue and Deleting It

```
Public Sub DeleteTestQueue()
    '=====
    'In this sub routine, we locate an pubic queue
    'in the Active Directory and delete it.
    '=====
    'Enable the error handler
    On Error GoTo DeleteTestQueue_Err
    'Declare variables for MSMQ objects.
    Dim oQuery      As MSMQ.MSMQQuery
    Dim oQInfos     As MSMQ.MSMQQueueInfos
    Dim oQInfo      As MSMQ.MSMQQueueInfo
    Dim oQueue      As MSMQ.MSMQQueue

    'Get MSMQQueueInfo objects and search for
    'the TestingQueue.
    Set oQuery = New MSMQ.MSMQQuery.
    Set oqinfor = oQuery.LookupQueue(Label:="TestingQueue")
    'Get the first MSMQQueueInfo object.
    Set oQInfo = oQInfos.Next

    'If the queue is not found, report it and get out.
    If oQInfo Is Nothing Then
        MsgBox "TestingQueue is not found!"
        Exit Sub
    End If

    'Delete the TestingQueue queue.
    oQInfo.Delete

    'If everything is ok, we are out of here.
    MsgBox "The queue is deleted!"
    Exit Sub
DeleteTestQueue_Err:
    MsgBox Err.Description
End Sub
```

In Listing 3.2, you used the Receive method to read the message and remove it from the queue. In Listing 3.4, you will use another technique to read the message selectively and remove only certain messages that meet certain criteria. Before you test the code in Listing 3.3, though, send two messages to the queue. Send the first message by running the code in Listing 3.1 without any modification. Then add `.AppSpecific = 25` to Listing 3.1 between the line `.Priority = 5` 'Default priority is 3 and

the line `.Body = "Testing Message"`. The code should now read as shown in the following segment:

```
Public Sub SendQueueMessage()
    '=====
    'In this sub routine, we will create a queue, open
    'the queue and send a testing message to the queue.
    '=====
    'Code is omitted here, see listing 3.1 for details.
    ' . . . . .
    'Prepare the message and send to the queue.
    With oMessage
        .Label = "Testing Message"
        .Priority = 5 'Default priority is 3.
        .AppSpecific = 25
        .Body = "Testing Message"
        .Send oQueue
    End With
    'The rest of the code is omitted, see Figure 3.1.
End Sub
```

Then run the modified code, and a message with the `AppSpecific` property set to 25 is sent to the queue. Figure 3.15 shows the two messages sent to the queue.

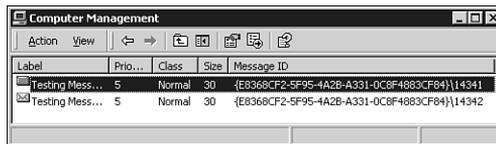


Figure 3.15: Two messages in the queue.



EXAMPLE

Listing 3.4 uses *Peek* methods (`PeekCurrent` and `PeekNext`) to search the queue for specific messages that meet certain criteria without removing them. If a specific message is found, the code will remove the message from the queue using the `ReceiveCurrent` method and also print the label and body of the message in the Debug window.

Listing 3.4 Searching for Specific Messages to Remove from the Queue

```
Public Sub FilterMessages()
    '=====
    'In this sub routine, we open an existing queue
    'and selectively retrieve a message.
    '=====
    'Enable the error handler
```

Listing 3.4 continued

```

On Error GoTo FilterMessages_Err
'Declare variables for MSMQ objects.
Dim oQInfo      As MSMQ.MSMQQueueInfo
Dim oQueue      As MSMQ.MSMQQueue
Dim oMessage    As MSMQ.MSMQMessage

'Initialize the MSMQQueueInfo object.
Set oQInfo = New MSMQQueueInfo
'we use a conditional compilation constant
'to take care of both public and private queues.
#If bUseDS Then
    oQInfo.PathName = ".\TestingQueue"
#Else
    oQInfo.PathName = ".\PRIVATE$\TestQueue"
#End If
'Open the queue for receive access while deny shared receive.
Set oQueue = oQInfo.Open(MQ_RECEIVE_ACCESS, MQ_DENY_RECEIVE_SHARE)
'If the queue is opened successfully,
'we process the messages.
If oQueue.IsOpen Then
    'Peek at the first message in the queue.
    Set oMessage = oQueue.PeekCurrent(ReceiveTimeout:=1000)
    'Search for specific messages with AppSpecific set to 25.
    'If found, Retrieve the message and print it.
    Do Until oMessage Is Nothing
        If oMessage.AppSpecific = 25 Then
            Set oMessage =
oQueue.ReceiveCurrent(ReceiveTimeout:=1000)
            Debug.Print "Message Label: " & oMessage.Label & vbCrLf
            Debug.Print "Message Body: " & oMessage.Body
            'Keep searching.
            Set oMessage = oQueue.PeekCurrent(ReceiveTimeout:=1000)
        Else
            Set oMessage = oQueue.PeekNext
        End If
    Loop
Else
    'Queue is not open, report the error and get out.
    MsgBox "The queue is not open!"
    Exit Sub
End If
'If everything is ok, we are out of here.
Exit Sub
FilterMessages_Err:
    MsgBox Err.Description
End Sub

```

After executing the code in Listing 3.4, you get results similar to those shown in Figure 3.14. If you open the Computer Management snap-in, you will notice that the second message you saw in Figure 3.15 is gone, as you can see in Figure 3.16.

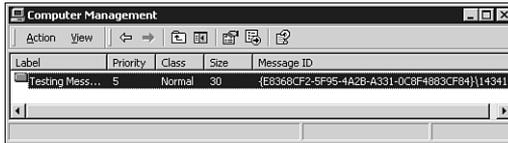


Figure 3.16: The message with `AppSpecific = 25` is removed from the queue.

Listing 3.4 filters messages based on the `AppSpecific` property. You can also use other message properties to look for specific messages. For example, you can use the `MsgClass` property to filter out report messages. To do so, simply change the line `.AppSpecific = 25` in Listing 3.4 to

```
.MsgClass = MQMSG_CLASS_REPORT
```

Advanced MSMQ Techniques

In this section, you will look at some more advanced MSMQ techniques. The first example demonstrates how to use the `MSMQEvent` object to retrieve messages asynchronously. In this example, you will create two Visual Basic applications: one to act as a message sender and another to act as a message receiver, as illustrated in Figure 3.17.

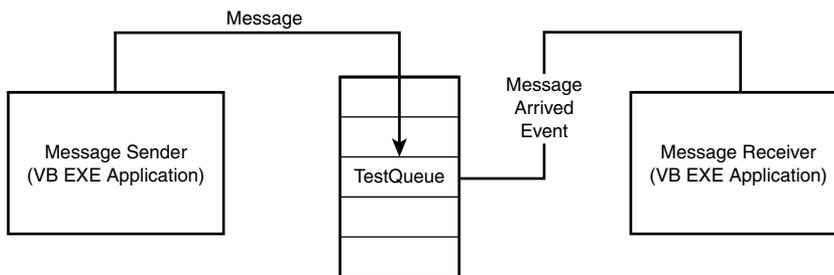


Figure 3.17: An MSMQ event example.

The Message Sender application in Figure 3.17 is a standard Visual Basic EXE project that contains a single form with a text box and a command button (see Figure 3.18).

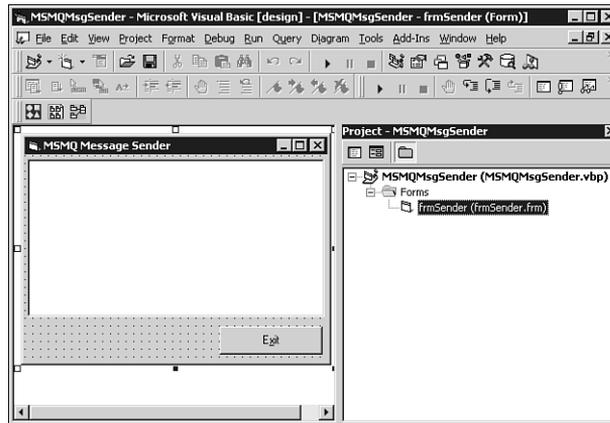


Figure 3.18: The MSMQMsgSender Visual Basic project.

The MultiLine property of the text box is better set to True so that it will function more like a text editor.

Listing 3.5 contains the code for the Message Sender application.



EXAMPLE

Listing 3.5 The MSMQMsgSender Project

```
'=====
'This is a sample MSMQ message sender application.
'It is paired with another MSMQ Receiver
'application to demonstrate how MSMQ event works.
'=====
Option Explicit

'=====
'The Change event of the text box tracks your key
'stroke and sends a message to the TestQueue every
'time when you press a key on the keyboard
'=====
Private Sub txtMessage_Change()
    'Enable the error handler
    On Error GoTo MessageSend_Error
    'Declare variables for MSMQ objects.
    Dim oQInfo As New MSMQ.MSMQQueueInfo
    Dim oQMsg As New MSMQ.MSMQMessage
    Dim oQueue As MSMQ.MSMQQueue

    'Set the path name to the TestQueue.
    #If bUseDS Then
        oQInfo.PathName = ".\TestQueue"
```

Listing 3.5 continued

```

#Else
    oQInfo.PathName = ".\PRIVATE$\TestQueue"
#End If

'Open the queue for send access.
Set oQueue = oQInfo.Open(MQ_SEND_ACCESS, MQ_DENY_NONE)

'Prepare the message and send the queue.
With oQMsg
    .Label = "MSMQ Event Testing"
    .Body = txtMessage.Text
    .Send oQueue
End With
'If everything is ok, close the queue and get out.
oQueue.Close
Exit Sub
MessageSend_Error:
    MsgBox Err.Description
End Sub

'=====
'The Click event of the Exit button.
'=====
Private Sub cmdExit_Click()
    'Exit the program.
    Unload Me
End Sub

```

The code in Listing 3.5 is very straightforward. In the `txtMessage_Change()` event of the text box, you put some code to send the content of the text box as a message to the `TestQueue` created in previous sections.

The Message Receiver application in Figure 3.17 is another Standard Visual Basic EXE project that has a single form with a text box and command button on it. It looks similar to the Message Sender application with the text box grayed out and locked to prevent editing (see Figure 3.19).

The size of each MSMQ message is limited to 4MB. As you learned earlier, however, the data type of the message can be almost anything. In the next example, you will create a disconnected ADO recordset from the database and send the recordset as a message to the queue. Later, you'll retrieve the message (ADO recordset) from the queue and display its content in the Visual Basic debug window. For details about ADO programming, see Chapter 2, "Windows DNA 2000 and COM+."

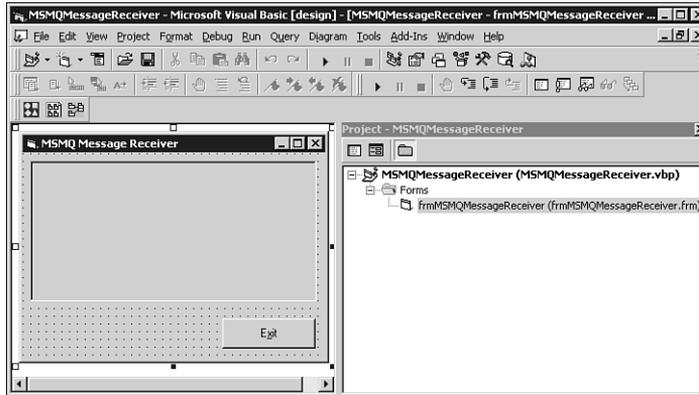


Figure 3.19: The MSMQMessageReceiver Visual Basic project.



EXAMPLE

Listing 3.6 shows the code for the Message Receiver application.

Listing 3.6 The MSMQMessageReceiver Project

```
'=====
'This is a sample MSMQ message receiver application.
'It is paired with the MSMQ Sender
'application to demonstrate how MSMQ event works.
'=====

Option Explicit
'Declare some model level variables for MSMQ objects.
Dim oQInfo As New MSMQ.MSMQQueueInfo
Dim oQReceive As MSMQ.MSMQQueue
Dim WithEvents oQEvent As MSMQ.MSMQEvent

'=====
'The form load event then opens the
'TestQueue and enables event notification.
'=====

Private Sub Form_Load()
    'Enable error handler.
    On Error GoTo Load_Err
    'Set the PathName of the queue.
    #If bUseDS Then
        oQInfo.PathName = ".\TestQueue"
    #Else
        oQInfo.PathName = ".\PRIVATE$\TestQueue"
    #End If
    'Open the queue for receive access.
    Set oQReceive = oQInfo.Open(MQ_RECEIVE_ACCESS, MQ_DENY_NONE)
    'Set the MSMQEvent object.
```

Listing 3.6 continued

```

    Set oQEvent = New MSMQ.MSMQEvent
    'Enable MSMQ event notification.
    oQReceive.EnableNotification oQEvent
    Exit Sub
Load_Err:
    MsgBox Err.Description
End Sub

'=====
'The Click event of the Exit button.
'=====
Private Sub cmdExit_Click()
    'Exit the program.
    Unload Me
End Sub

'=====
'The Arrived event of the MSMQEvent object.
'Whenever this event fires, we update the content
'of the text box. Remember to enable the event
'notification for ensuring the firing of the
'subsequent events.
'=====
Private Sub oQEvent_Arrived(ByVal Queue As Object, _
                           ByVal Cursor As Long)
    'Enable error handler.
    On Error GoTo Event_Arrived_Err
    'Declare the MSMQMessage object.
    Dim oQMsg As MSMQ.MSMQMessage
    'Retrieve the message and display its contents in the text box.
    Set oQMsg = oQReceive.ReceiveCurrent(ReceiveTimeout:=1000)
    txtMessage = oQMsg.Body
    'Important!!!--Enable event notification before exiting the event.
    oQReceive.EnableNotification Event:=oQEvent, Cursor:=MQMSG_FIRST
    Exit Sub
Event_Arrived_Err:
    MsgBox Err.Description
End Sub

'=====
'The ArrivedError event of MSMQEvent object.
'This event will be fired when the EnableNotification
'of the message object is called and an error has
'been generated. The ErrorCode is the return code
'of the ReceiveCurrent call of the MSMQQueue object.
'=====

```

Listing 3.6 continued

```
Private Sub oQEvent_ArrivedError(ByVal Queue As Object, _  
                                ByVal ErrorCode As Long, _  
                                ByVal Cursor As Long)  
    MsgBox "Error event fired!" & vbCrLf & _  
        "Error: " & Hex(ErrorCode)  
End Sub
```

In Listing 3.6, the Load event of the form opens the queue, initializes the event object, and enables event notification. The Arrived event receives the message, updates the content of the text box with the message, and enables event notification before you exit the event procedure. To see how this listing works, run two separate instances of the Message Sender and the Message Receiver applications. Arrange the screens so that you can see both of them at the same time. Notice that whenever you type something in the text box of the Send application, its content also appears in the text box of the Receiver application, as shown in Figure 3.20.



Figure 3.20: An MSMQ event in action.

The event notification capability of MSMQ enables you to develop some very powerful applications that are event-driven rather than message pulling (such as frequently checking the message according to a predefined time interval).

The next example demonstrates another powerful feature of MSMQ: sending an ADO recordset as a message. In this example, you will use a simple Visual Basic form with two command buttons: cmdSendRecordset and cmdReadRecordset (see Figure 3.21).



EXAMPLE

In the click event of cmdSendRecordset, you will create a disconnected recordset with six programming titles from the pubs database of SQL Server and send the recordset as a message to the TestQueue created earlier. In the click event of the cmdReadRecordset, you will receive the message of the recordset and display its contents in the debug window. Listing 3.7 illustrates the code for this example.

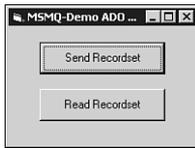


Figure 3.21: An MSMQ ADO recordset example.

Listing 3.7 ADO Recordset as the MSMQ Message

```
'=====
'In this example, we demonstrate how to send
'a disconnected recordset as a MSMQ message.
'=====

Option Explicit

Private Sub cmdSendRecordset_Click()
    'Enable the error handler.
    On Error GoTo SendRecordset_Err

    'Declare variables.
    Dim rsTitles As New ADODB.Recordset
    Dim oQinfo As New MSMQ.MSMQQueueInfo
    Dim oQueue As MSMQ.MSMQQueue
    Dim oMessage As New MSMQ.MSMQMessage
    Dim sConnection As String
    Dim sSQL As String

    'Set connection string and SQL statement.
    sConnection = "pubs"
    sSQL = "select title from titles where title_id like 'BU%'"

    'Create a disconnected recordset.
    With rsTitles
        .CursorLocation = adUseClient
        .CursorType = adOpenStatic
        .LockType = adLockBatchOptimistic
        .Open sSQL, sConnection
    End With

    'Set the PathName of the MSMQQueueInfo object.
    #If bUseDS Then
        oQinfo.PathName = ".\TestQueue"
    #Else
```

Listing 3.7 continued

```
        oQinfo.PathName = ".\PRIVATE$\TestQueue"
    #End If

    'Open the queue for send access.
    Set oQueue = oQinfo.Open(MQ_SEND_ACCESS, MQ_DENY_NONE)

    'Send the ADO recordset to the queue.
    With oMessage
        .Label = "ADO recordset"
        .Body = rsTitles
        .Send oQueue
    End With

    'If everything is okay, clean up and get out of here.
    oQueue.Close
    rsTitles.Close

    MsgBox "Recordset sent!"
    Exit Sub
SendRecordset_Err:
    MsgBox Err.Description
End Sub

Private Sub cmdReadRecordset_Click()
    'Enable the error handler.
    On Error GoTo ReadRecordset_Err

    'Declare object variables.
    Dim rsTitles As ADODB.Recordset
    Dim oQinfo As New MSMQ.MSMQQueueInfo
    Dim oQueue As MSMQ.MSMQQueue
    Dim oMessage As MSMQ.MSMQMessage

    'Set the PathName of the MSMQQueueInfo object.
    #If bUseDS Then
        oQinfo.PathName = ".\TestQueue"
    #Else
        oQinfo.PathName = ".\PRIVATE$\TestQueue"
    #End If

    'Open the queue for read access.
    Set oQueue = oQinfo.Open(MQ_RECEIVE_ACCESS, MQ_DENY_NONE)

    'Read the message.
    Set oMessage = oQueue.Receive(ReceiveTimeout:=1000)
```

Listing 3.7 continued

```

If Not oMessage Is Nothing Then
    'Assign the message body to an ADO recordset.
    Set rsTitles = oMessage.Body
    'Loop through the recordset and display its contents.
    Do Until rsTitles.EOF
        Debug.Print rsTitles("title")
        rsTitles.MoveNext
    Loop
    rsTitles.Close
End If
oQueue.Close
'If everything is okay, we are out of there.
Exit Sub
ReadRecordset_Err:
    MsgBox Err.Description
End Sub

```

Run this example, and click the Send Recordset button. A disconnected ADO recordset is then placed on the TestQueue (see Figure 3.22).



Figure 3.22: The ADO recordset is put in the queue.

NOTE

The size of the message on your machine may be a little different from the size you saw in Figure 3.22.

If you then click the Read Recordset button, the recordset is dequeued, and its contents are listed in the debug window (see Figure 3.23).

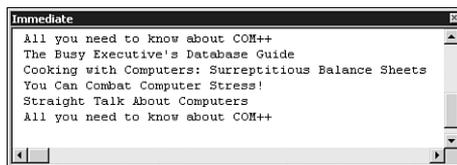


Figure 3.23: The content of the recordset in the debug window.

Creating a disconnected ADO recordset is a very efficient means by which you can pass data between different tiers in DNA applications. With this

technique, combined with the asynchronous processing power of MSMQ, you can build more scalable and robust enterprise and Internet applications.

An Asynchronous Ordering Application

So far, I have introduced all MSMQ programming techniques in Visual Basic. In this section, you will use a more complicated example, an asynchronous ordering system, to learn how to use MSMQ in the real world.

Figure 3.24 illustrates the workflow of this ordering system. An ordering application sends the order data to OrderQueue as a message (step 1), which specifies OrderResponseQueue as the response queue (step 2). When the order message arrives in the OrderQueue, an event fires in the order processing application (step 3), which in turn inserts the order into the Orders table in the database by calling a stored procedure (step 4). When the order processing application finishes processing, it sends a confirmation message back to the OrderResponseQueue (step 5). When the confirmation message arrives in the OrderResponseQueue, an event fires and the results are displayed (step 6).

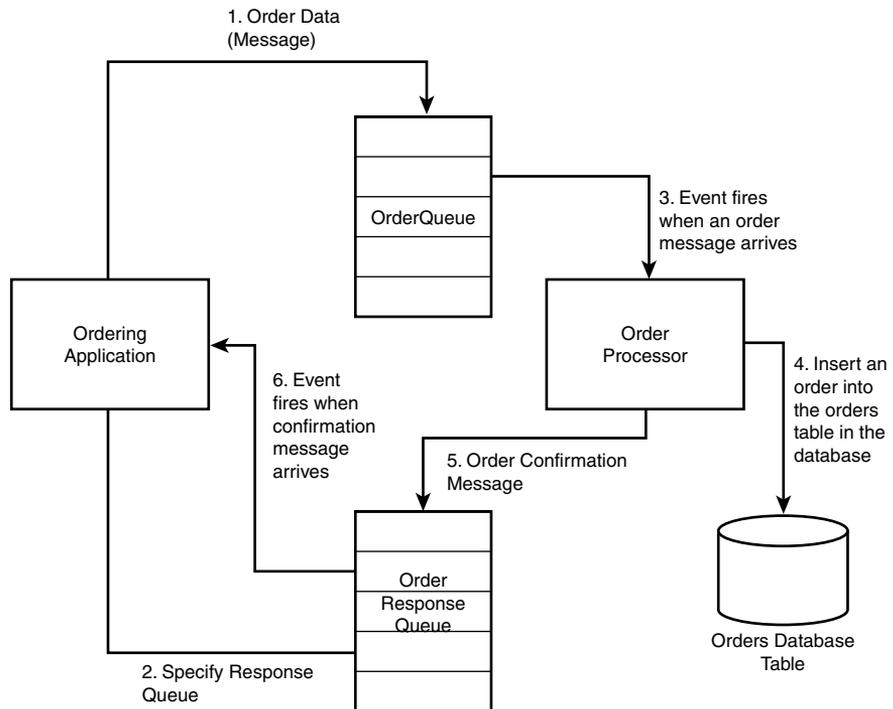


Figure 3.24: The workflow of the asynchronous ordering system.



EXAMPLE

The purpose of this example is to demonstrate how to leverage the asynchronous processing power of MSMQ to build highly scalable and robust applications. You will use the Orders table in the Northwind database that comes with SQL Server 7.0 in this example. For the sake of simplicity, you can ignore the Order Details table. To follow this example in Listing 3.8 and Listing 3.9, you need to create a system DSN named Northwind, which points to the Northwind database. You also need to create a stored procedure that inserts a row in the Orders table and returns the current OrderID as an output parameter.

Listing 3.8 Stored Procedure PlaceOrder

```
Use Northwind
go

if exists (select * from sysobjects where id = object_id('PlaceOrder'))
    drop proc PlaceOrder
go

create proc PlaceOrder
    @Order varchar(300),
    @OrderID int out
as
declare @sql varchar(600)

select @sql= 'insert Orders ('
           + 'CustomerID, '
           + 'EmployeeID, '
           + 'OrderDate,'
           + 'RequiredDate,'
           + 'ShippedDate,'
           + 'ShipVia,'
           + 'Freight,'
           + 'ShipName,'
           + 'ShipAddress,'
           + 'ShipCity,'
           + 'ShipPostalCode,'
           + 'ShipCountry'
           + ') values (' + @Order + ')'

--Insert the order to the Orders table.
exec(@sql)
--Return the OrderID for the newly inserted order.
select @OrderID = max(OrderID) from Orders
go
```

You can use the Computer Management snap-in to create the two queues for this example: the OrderQueue and the OrderResponseQueue (see Figure 3.25).

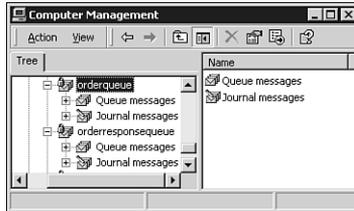


Figure 3.25: The OrderQueue and the OrderResponseQueue.

Figure 3.26 shows the asynchronous ordering system with the ordering application on the left and the order processing application on the right.

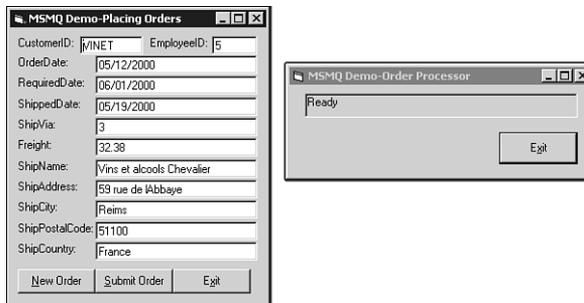


Figure 3.26: The asynchronous ordering system.

When you fill up the order information on the form and click the Submit Order button, the status of the ordering processing application briefly changes to Processing order and then back to Ready. Depending on the CPU speed and how much RAM you have on your machine, you may hardly notice the status change. Soon a message box pops up and confirms that your order (with an OrderID) is processed (see Figure 3.27).

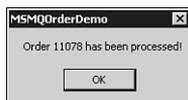


Figure 3.27: The confirmation message of the asynchronous ordering system.



EXAMPLE

Listings 3.9 and 3.10 provide the complete code for this application and reveal what's happening behind the scenes.

Listing 3.9 The Ordering Application

```

'=====
'The ordering application of the
'asynchronous ordering system
'=====

'=====
'General Declarations section
'=====

Option Explicit
'Declare module level variables.
Dim oQinfoOrder          As New MSMQ.MSMQQueueInfo
Dim oQInfoOrderResponse As New MSMQ.MSMQQueueInfo
Dim oQueueResponse      As MSMQ.MSMQQueue
Dim WithEvents oQEvent  As MSMQ.MSMQEvent

'=====
'The Load event of the form
'=====
Private Sub Form_Load()
    'In the load event of the form, specify PathNames for
    'both OrderQueue and OrderResponseQueue.
    On Error GoTo Load_Err
    #If bUseDS Then
        oQinfoOrder.PathName = ".\OrderQueue"
        oQInfoOrderResponse.PathName = ".\OrderResponseQueue"
    #Else
        oQinfoOrder.PathName = ".\PRIVATES\OrderQueue"
        oQInfoOrderResponse.PathName = ".\PRIVATES\OrderResponseQueue"
    #End If

    'Open the OrderResponseQueue and prepare to receive events.
    Set oQueueResponse = oQInfoOrderResponse.Open(MQ_RECEIVE_ACCESS, _
                                                    MQ_DENY_NONE)

    Set oQEvent = New MSMQ.MSMQEvent

    'Enable message notification.
    oQueueResponse.EnableNotification oQEvent
    Exit Sub
Load_Err:
    MsgBox Err.Description
End Sub

'=====
'The Click event of the New Order button
'=====

```

Listing 3.9 continued

```

Private Sub cmdNewOrder_Click()
    'Clear all input boxes.
    Dim oControl As Control
    For Each oControl In Me.Controls
        If TypeOf oControl Is TextBox Then
            oControl.Text = ""
        End If
    Next oControl
End Sub

'=====
'The Click event of the Submit Order button
'=====
Private Sub cmdSubmit_Click()
    On Error GoTo SubmitOrder_Err
    Dim oQueue           As MSMQ.MSMQQueue
    Dim oMessage         As New MSMQ.MSMQMessage
    Dim sMessage         As String

    'Simple client side data validation.
    If Len(txtCustomerID) + _
        Len(txtEmployeeID) + _
        Len(txtOrderDate) + _
        Len(txtRequiredDate) + _
        Len(txtShipDate) + _
        Len(txtShipVia) + _
        Len(txtFreight) + _
        Len(txtShipName) + _
        Len(txtShipAddress) + _
        Len(txtShipCity) + _
        Len(txtShipPostalCode) + _
        Len(txtShipCountry) = 0 Then

        MsgBox "Incomplete order!", vbCritical
        Exit Sub
    End If

    'Gather information from the order form
    'and pad them into a message.
    sMessage = "" & txtCustomerID & ", " _
        & txtEmployeeID & ", " _
        & "" & txtOrderDate & ", " _
        & "" & txtRequiredDate & ", " _
        & "" & txtShipDate & ", " _
        & txtShipVia & ", " _
        & txtFreight & ", " _

```

Listing 3.9 continued

```

        & "" & txtShipName & ", " _
        & "" & txtShipAddress & ", " _
        & "" & txtShipCity & ", " _
        & "" & txtShipPostalCode & ", " _
        & "" & txtShipCountry & ""

Screen.MousePointer = vbHourglass

'Open the OrderQueue for send access and send the order
'message to the queue.
Set oQueue = oQInfoOrder.Open(MQ_SEND_ACCESS, MQ_DENY_NONE)
sMessage = sMessage
With oMessage
    .Label = "Order"
    .Body = sMessage
    'Specify the response queue.
    Set .ResponseQueueInfo = oQInfoOrderResponse
    .Send oQueue
End With
oQueue.Close
Screen.MousePointer = vbDefault

Exit Sub

SubmitOrder_Err:
    Screen.MousePointer = vbDefault
    MsgBox Err.Description
End Sub

'=====
'The Arrived event of the OrderResponseQueue
'=====
Private Sub oQEvent_Arrived(ByVal Queue As Object, ByVal Cursor As Long)
    'Display the response message when it arrives.
    On Error GoTo Event_Arrived_Err

    Dim oMessage As New MSMQ.MSMQMessage
    Set oMessage = oQueueResponse.ReceiveCurrent(ReceiveTimeout:=1000)
    MsgBox oMessage.Body

    'Enable message notification before exiting the event.
    oQueueResponse.EnableNotification oQEvent

Exit Sub
Event_Arrived_Err:

```

Listing 3.9 continued

```

    MsgBox Err.Description
End Sub

'=====
'The ArrivedError event of the OrderResponseQueue
'=====
Private Sub oQEvent_ArrivedError(ByVal Queue As Object, _
                                ByVal ErrorCode As Long, _
                                ByVal Cursor As Long)
    MsgBox "Error event fired!" & vbCrLf & _
        "Error: " & Hex(ErrorCode)
End Sub

'=====
'The Click event of the Exit button
'=====
Private Sub cmdExit_Click()
    Unload Me
End Sub

```

Listing 3.10 The Order Processing Application

```

'=====
'The order processing application of
'the asynchronous ordering system
'=====

'=====
'General Declarations section
'=====
Option Explicit
'Declare module level variables.
Dim oQinfoOrder          As New MSMQ.MSMQQueueInfo
Dim oQueue               As MSMQ.MSMQQueue
Dim WithEvents oQEvent   As MSMQ.MSMQEvent

'=====
'The Load event of the form
'=====
Private Sub Form_Load()
    'Listen to the event of the OrderQueue.
    #If bUseDS Then
        oQinfoOrder.PathName = ".\OrderQueue"
    #Else

```

Listing 3.10 continued

```

        oQinfoOrder.PathName = ".\PRIVATE$\OrderQueue"
    #End If

    Set oQueue = oQinfoOrder.Open(MQ_RECEIVE_ACCESS, MQ_DENY_NONE)
    Set oQEvent = New MSMQ.MSMQEvent

    lblStatus = "Ready"

    'Enable message notification.
    oQueue.EnableNotification oQEvent

End Sub

'=====
'The Arrived event of the OrderQueue
'=====
Private Sub oQEvent_Arrived(ByVal Queue As Object, ByVal Cursor As Long)
    'Process the order message when it arrives and
    'send a response message when the process is finished.
    On Error GoTo Event_Arrived_Err

    Dim oMessage          As New MSMQ.MSMQMessage
    Dim oQueueResponse    As MSMQ.MSMQQueue
    Dim oResponseMessage  As New MSMQ.MSMQMessage
    Dim oConnection       As New ADODB.Connection
    Dim oCommand           As New ADODB.Command
    Dim iOrderID           As Integer
    Dim sMessage           As String

    'Update the status.
    Screen.MousePointer = vbHourglass
    lblStatus = "Processing order..."
    DoEvents
    'Read the message.
    Set oMessage = oQueue.ReceiveCurrent(ReceiveTimeout:=1000)
    sMessage = oMessage.Body

    'Connect to the Northwind database.
    oConnection.Open "Northwind"

    'Call the stored procedure "PlaceOrder".
    With oCommand
        .ActiveConnection = oConnection
    End With

```

Listing 3.10 continued

```

        .CommandType = adCmdStoredProc
        .CommandText = "PlaceOrder"
        .Parameters.Append .CreateParameter("@Order", _
                                           adVarChar, _
                                           adParamInput, _
                                           300)
        .Parameters.Append .CreateParameter("@OrderID", _
                                           adInteger, _
                                           adParamOutput)

        .Parameters("@Order") = sMessage
        .Execute
        iOrderID = .Parameters("@OrderID")
    End With

    'If the response queue is specified then send a confirmation mes-
    sage.
    If Not oMessage.ResponseQueueInfo Is Nothing Then
        Set oQueueResponse = _
            oMessage.ResponseQueueInfo.Open(MQ_SEND_ACCESS,
MQ_DENY_NONE)
        With oResponseMessage
            .Label = "Order Confirmation Message"
            .Body = "Order " & CStr(iOrderID) & " has been processed!"
            .Send oQueueResponse
        End With
    End If
    lblStatus = "Ready"

    'Enable message notification.
    oQueue.EnableNotification oQEvent
    Screen.MousePointer = vbDefault

    Exit Sub
Event_Arrived_Err:
    Screen.MousePointer = vbDefault
    lblStatus = "Ready"
    MsgBox Err.Description
End Sub

'=====
'The ArrivedError event of the OrderQueue
'=====
Private Sub oQEvent_ArrivedError(ByVal Queue As Object, _

```

Listing 3.10 continued

```

ByVal ErrorCode As Long, _
ByVal Cursor As Long)

MsgBox "Error event fired!" & vbCrLf & _
      "Error: " & Hex(ErrorCode)

End Sub

'=====
'The Click event of the Exit button
'=====
Private Sub cmdExit_Click()
    Unload Me
End Sub

```

When the form of the ordering application is loaded, it establishes the pathnames for both `OrderQueue` and `OrderResponseQueue`, opens `OrderResponseQueue`, and enables the event for receiving order confirmation messages (refer to Listing 3.9). After you fill in the order form and click the `Submit Order` button, the click event packs the order into a string message and sends the message to the `OrderQueue`, specifying `OrderResponseQueue` as the response queue (refer to Listing 3.9). When the order processing application starts, the `Load` event of the form establishes a pathname for the `OrderQueue` and enables the event to receive ordering messages (refer to Listing 3.10). When an ordering message arrives, it triggers the `Arrived` event. The code in the event calls a stored procedure that inserts the order to the `Orders` table in the Northwind database and returns an order ID. Then a confirmation message is sent to `OrderResponseQueue` (refer to Listing 3.10), which in turn triggers the event of the ordering application to display the confirmation message (refer to Listing 3.9).

To better understand how the system works, run the applications in a slow motion mode. Stop the order process application if it is running. Then start the ordering application, fill in the form, and click the `Submit Order` button. If you look at both `OrderQueue` and `OrderResponseQueue` at this point, you will find that the order message you just sent stays in `OrderQueue`, whereas no messages appear in `OrderResponseQueue` (see Figure 3.28).

Now stop the ordering application and start the order processing application. If you check the queues, you will notice that the order message on `OrderQueue` is gone and a confirmation message appears in `OrderResponseQueue` (see Figure 3.29).

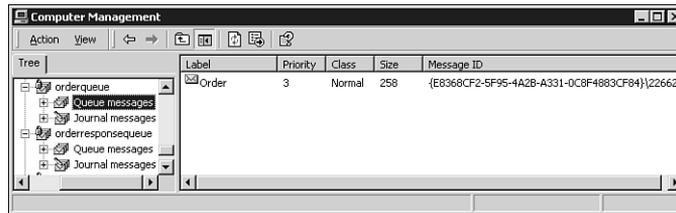


Figure 3.28: An order message in OrderQueue.

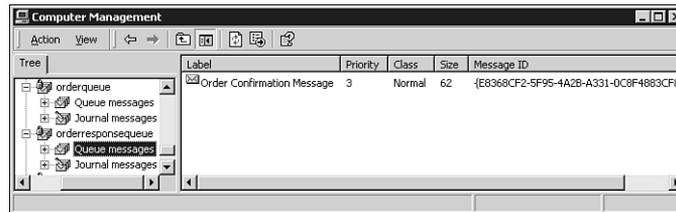


Figure 3.29: A confirmation message in OrderResponseQueue.

Now start the ordering application again. This time, you will see a confirmation message box. If you check the queues again, you will notice that no messages appear in OrderQueue or OrderResponseQueue.

What's Next

This chapter introduced MSMQ and showed you how to program MSMQ in Visual Basic. The knowledge you learned will be essential for you to understand important COM+ services, such as Queued Components (QC). In Chapter 4, "Introduction to Visual Basic COM Programming," you will learn how to develop COM components in Visual Basic.