

Microsoft Message Queue Service: Mitteilungen veröffentlichen und lesen

## Auf die Post ist Verlass

Programm A spricht, aber Programm B hat wichtigeres zu tun als zuzuhören. In solchen Fällen hilft der Microsoft Message Queue Service. Damit haben Entwickler eine fertige Infrastruktur für die asynchrone und zuverlässige Inter-Programmkommunikation an der Hand. Sie lässt sich für viele Anwendungsszenarien einsetzen. Diese dotnetpro-Serie stellt Ihnen MSMQ vor.

**W**as machen Sie, wenn Sie eine Frage an einen Kollegen haben, Sie ihn aber nicht bei der Arbeit stören wollen? Als moderner Mensch senden Sie ihm eine E-Mail. Danach widmen Sie sich erst einmal in Ruhe Ihren anderen Aufgaben. Ihr Kollege schaut später in seinen Posteingang und findet die Anfrage, löst das Problem natürlich in kürzester Zeit und möchte Ihnen am Telefon die Lösung mitteilen. Zu seinem Leidwesen erfährt er, dass Sie gerade nicht am Platz sind und sich in einer wichtigen Sitzung befinden. Also sendet er Ihnen die Antwort ebenfalls per E-Mail zurück.

### Auf einen Blick

#### Autor

**Marcel Gnoth** ist Leiter der Entwicklungsabteilung bei der Berliner NTeam GmbH. Neben seiner Arbeit als Programmierer führt er Trainings für Entwickler durch, schreibt für Fachmagazine und spricht regelmäßig auf Konferenzen. Seine Schwerpunkte liegen bei COM, Informationssystemen und der .NET-Programmierung. Sie erreichen ihn unter [mgnoth@nteam.de](mailto:mgnoth@nteam.de).



**dotnetpro.code**  
A0403MSMQ



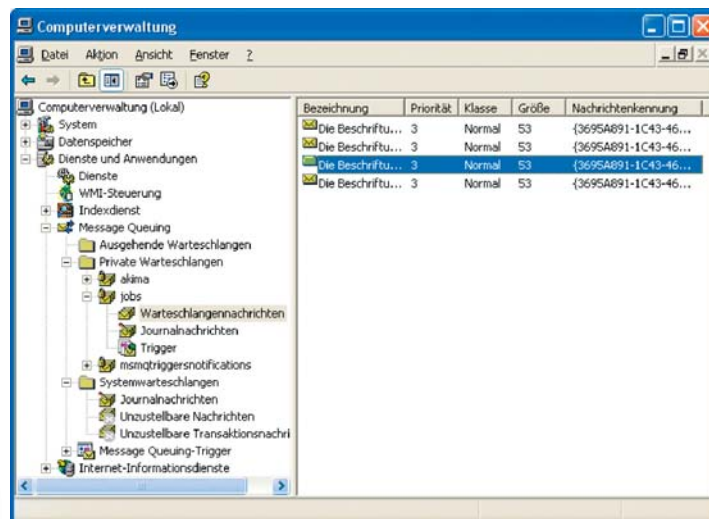
**Sprachen** Visual Basic.NET

**Technik** MSMQ

**Voraussetzungen** Windows XP oder Windows Server 2003, VS.NET

#### Serie

1. Microsoft Message Queue Service: Mitteilungen veröffentlichen, lesen
2. Microsoft Message Queue Service: HTTP und Trigger



**Abbildung 1**  
Über die Computerverwaltung können Queues und deren Nachrichten angesehen werden.

Dieses kleine Szenario ist ein typisches Beispiel für eine asynchrone Kommunikation. Eine Nachricht wird losgeschickt und der Empfänger holt sie sich, wenn er dafür bereit ist. In der Zwischenzeit kann der Sender an anderen Aufgaben weiterarbeiten. Bei einer synchronen Kommunikation, am Telefon zum Beispiel, wird der Empfänger in seiner Arbeit unterbrochen. Ist der Empfänger nicht erreichbar, dann hat der Sender bei einer synchronen Kommunikation ein Problem: Er muss warten, unter Umständen sehr lange.

### Programmbeispiele

Am Telefon sind Menschen ja meist noch geduldig und warten auch mal 60 Sekunden, ob der Angerufene den Hörer abnimmt, oder sie versuchen es mehrmals. Blockiert aber ein Computerprogramm, weil es auf Antwort wartet, drücken die Anwender schnell die [Strg]+[Alt]+[Entf]-Tasten und schießen das Programm einfach ab.

Ein Außendienstmitarbeiter möchte bereits unterwegs die Daten seiner Verkaufserfolge in das System eingeben, weil

er während der langen Bahnfahrt Zeit hat. Er tippt also alle Daten ein, aber erst, wenn der Laptop wieder ins heimische Firmennetz eingebunden ist, werden die Informationen in die Unternehmensdatenbank eingetragen oder gar irgendwelche Geschäftsprozesse angestoßen.

So ein Geschäftsprozess könnte eine Bestellung sein, die von mehreren Abteilungen genehmigt werden muss, ein Workflow, an dem mehrere Personen und Systeme beteiligt sind, die nicht immer gleich verfügbar sind. Da wäre es doch schön, wenn jeder Beteiligte ein Postfach hätte, wo solche Anfragen landen, und wenn alle zugestimmt haben oder einer abgelehnt hat, bekommt der Initiator eine Antwort. Dieser Vorgang kann sich schon mal über mehrere Tage erstrecken.

Ein Dienstleister mit einem vollen Terminkalender muss plötzlich alle Termine der nächsten drei Stunden absagen, weil irgendwas dazwischen gekommen ist. Es bleibt keine Zeit, alle Betroffenen, die im Kalender stehen, anzurufen und zu warten, bis diese ans Telefon gehen. Eine kurze E-Mail an alle, die im Kalender stehen, und der Dienstleister kann sich dem

Notfall zuwenden. Die Betroffenen sehen die Änderung augenblicklich in ihrem Kalender und brauchen zu dem Termin gar nicht erst zu erscheinen.

## E-Mail für Computerprogramme

Für alle oben genannten Beispiele stellt Ihnen Microsoft Message Queuing eine umfangreiche Infrastruktur zur Verfügung, mit der Sie Ihre Anwendungen umsetzen können. Jeder Computer kann über mehrere Queues (Nachrichtenwarteschlangen) verfügen. Zwischen diesen Warteschlangen können Nachrichten, ja ganze Objektbäume ausgetauscht werden. Dafür stehen Ihnen viele Optionen zur Verfügung. Sie können Bestätigungen anfordern, Nachrichten innerhalb einer Transaktion versenden oder den Versand protokollieren. Auch an Verschlüsselung und Zertifikate ist gedacht.

## Die Beispielanwendung

Die Beispielanwendung auf der Heft-CD ist ein einfacher Sender und Empfänger. Sie können die Anwendung mehrfach lokal oder auf anderen Rechnern starten. In die Textboxen tragen Sie die Queue-Pfade ein. So können Sie mit einem Client Nachrichten versenden und mit einem zweiten die Nachrichten empfangen. Die Anwendung demonstriert die hier im Artikel beschriebenen Techniken.

Achtung: Falls es zu Problemen mit dem Senden oder Empfangen kommt, prüfen Sie über die Computerverwaltung (Abbildung 1) die Berechtigung für die Queues.

## Was ist Microsoft Message Queuing?

Eingeführt wurde MSMQ 1998 als Bestandteil des NT-Option-Packs. Inzwischen ist es fester Bestandteil des Windows-Setups und wird auch über die Windows Service Packs aktualisiert. MSMQ ist für viele Versionen der Windowsfamilie verfügbar. Das geht von Windows 95 (NT-Option Pack) und NT4 über Windows 2000 und Windows XP bis hin zu Windows CE 3.0.

Die Nachrichtenwarteschlangen eines Computers werden von einem Windows-Dienst verwaltet. Dieser Dienst wird standardmäßig nicht mitinstalliert. An der Kommandozeile können Sie mit `net start msmq` oder `net stop msmq` den Dienst starten oder beenden. Über die Compu-

## Objektpersistenz.

Ein Objekt wird auf Computer A instanziiert, Eigenschaften des Objektes werden verändert. Dann wird das Objekt eingepackt, zu Computer B transportiert und wieder ausgepackt. Dort kann ein anderer Prozess etwas mit dem Objekt tun. Anschließend wird das Objekt wieder verpackt und zurückgesendet.

Oder das Objekt wird verpackt, auf der Festplatte gespeichert, der Computer heruntergefahren. Nach dem Neustart wird das Objekt von der Festplatte gelesen und wieder ausgepackt, die Arbeit kann weitergehen, als wäre nichts gewesen.

Um ein Objekt einzupacken, müssen alle wichtigen Eigenschaften in einen Puffer geschrieben werden, der dann als Folge von Bytes übertragen oder gespeichert werden kann. Auf der Empfängerseite muss es dann möglich sein, mithilfe des übertragenen Puffers ein neues leeres Objekt so zu füllen, dass alle „wichtigen“ Eigenschaften wieder so gesetzt sind wie bei dem Originalobjekt. So wird eine Kopie des Objektes erzeugt.

Was dabei eine wichtige Eigenschaft ist, bestimmt die Semantik der Anwendung. Stammdaten, wie Name, Adresse oder auch Datenbank-ID werden sicher immer eingepackt werden, eine Thread-ID sicher nicht, da sie bei beiden instanziierten Objekten, Original und Kopie unterschiedlich sein wird.

Das Ein- und Auspacken übernehmen so genannte *Formatter* in .NET. Ihnen stehen zwei zur Verfügung: der *BinaryFormatter* und der *SOAPFormatter*. Als Puffer wird ein Stream-Objekt verwendet, dafür stehen Ihnen auch mehrere zur Verfügung, wie *MemoryStream*, *FileStream*, *NetworkStream*.

Einem Formatter übergeben Sie ein Objekt und einen Stream und der Formatter schreibt dann das Objekt in den Stream rein. Über den Reflection Mechanismus kann der Formatter das Objekt untersuchen und dann alle Eigenschaftswerte in den Stream schreiben.

Sie brauchen Ihre Klassen nur mit dem *Serializable*-Attribut zu kennzeichnen, für den Rest wird gesorgt. Oder Sie implementieren das *ISerializable*-Interface in Ihrer Klasse und steuern den Serialisierungsvorgang manuell.

MSMQ verfügt über eigene Formatter, die den Framework-Formattern ähnlich sind. Es gibt Binär-, XML- und ActiveX-Formatter. Über MSMQ können auch COM-Komponenten versendet werden, der *ActiveXMessageFormatter* ermöglicht die Zusammenarbeit mit VB6-Applikationen.

Der Formatter wird über die *MessageQueue.Formatter*-oder *Message.Formatter*-Property eingestellt.

terverwaltung (siehe Abbildung 1) können Sie MSMQ administrieren.

## MSMQ-Modus

Bei der Installation müssen Sie sich für einen der beiden Modi entscheiden, in dem MSMQ betrieben werden soll:

- Active Directory (NT4: MQIS)
- Workgroup (Arbeitsgruppe)

Arbeitet MSMQ im Active-Directory-Modus, dann werden alle Informationen über Warteschlangen des lokalen Computers in einem zentralen Informationsverzeichnis eingetragen. Der Vorteil liegt darin, dass andere Computer eine Warteschlange im Informationsverzeichnis suchen können und dann die Adresse der Warteschlange erhalten. So können wich-

tige Warteschlangen immer gefunden werden, auch wenn sie auf einen anderen Computer „umgezogen“ sind.

Der Zugriff auf eine solche Warteschlange ist allerdings immer ein bisschen langsamer, da ja ihre Adresse erst im Informationsverzeichnis nachgeschlagen werden muss. Bei der Installation von MSMQ im Active-Directory-Modus werden auch entsprechende Berechtigungen für die Eintragung in das Informationsverzeichnis benötigt.

Dieser Weg ist nicht immer praktikabel und so kann MSMQ auch im Workgroup-Modus betrieben werden. Gerade, wenn Sie Ihre ersten Versuche mit MSMQ machen, werden Sie nicht immer ein Active Directory zur Verfügung haben. Die meisten Funktionen können auch im Workgroup-Modus verwendet werden. Die In-

Installation ist wesentlich einfacher und Sie benötigen nur Berechtigungen auf der lokalen Maschine.

Das Active Directory wurde erst mit Windows 2000 eingeführt. Unter NT4 musste auf einem Server ein sogenannter *Primary Enterprise Controller (PEC)*, der einen lokalen SQL-Server benötigte, installiert werden. Dieser PEC verwaltete dann ein eigenes Informationsverzeichnis, den *Message Queue Information Store (MQIS)*. Bei der Installation eines Clients musste ein MQIS angegeben werden.

Es gibt auch noch andere Rollen für einen MSMQ-Computer. So können alle MSMQ-Clients eines LANs von einem *Site Controller* verwaltet werden, der Nachrichten zwischen LANs routet. Eine ausführliche Beschreibung von MSMQ und NT 4 finden Sie in [1].

### Installation

Wenn MSMQ mit einem Verzeichnisdienst betrieben werden soll, dann muss MSMQ als Erstes auf dem Active Directory Controller installiert werden. Dieser Server wird dann bei den Client-Installationen als Server angegeben. Hinweise zur Installation unter NT 4 finden Sie unter [1].

Bei einer Standard-Windows-Installation wird MSMQ nicht mitinstalliert, Sie müssen dafür die *Systemsteuerung/Software/Windows Komponenten* aufrufen und dort MSMQ auswählen (siehe Abbildung 2). Unter Windows XP wählen Sie Details für die Installation von MSMQ aus (siehe Abbildung 3). Dort müssen Sie *Integration des Active Directory* auswählen um mit einem Verzeichnisdienst zu arbeiten. Wenn Sie diese Option auswählen, dann erscheint während der Installation der Dialog aus Abbildung 4.

Auf die Optionen *HTTP* und *Trigger* wird der zweite Teil dieser Serie eingehen.

Es gibt zwei verschiedene Optionen für einen MSMQ-Client: unabhängig (independent) und abhängig (dependent). Die Auswahl treffen Sie, indem Sie bei der Option *Allgemein* auf *Details* klicken. Dort wird Ihnen *Basisfunktionalität* und *Lokaler Speicher* angeboten. Wählen Sie nur *Basisfunktionalität*, dann bekommen Sie einen *dependent* Client, der immer eine Verbindung zu einem Server benötigt, da er lokal keine Warteschlangen und Nachrichten zwischenspeichert.

Ein *dependent* Client ist in seiner Funktionalität sehr eingeschränkt und sollte nur verwendet werden, wenn ein independent Client nicht in Frage

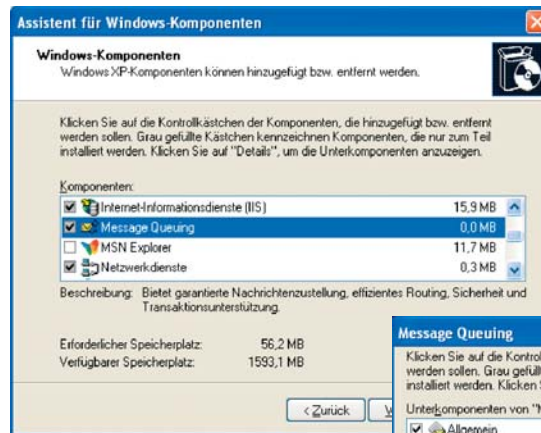


Abbildung 2 Das Windows Setup.

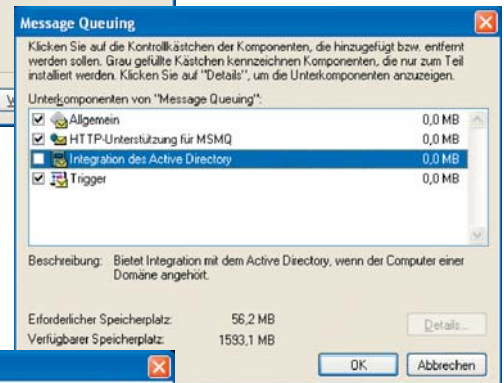


Abbildung 3 MSMQ-Setup-Details.

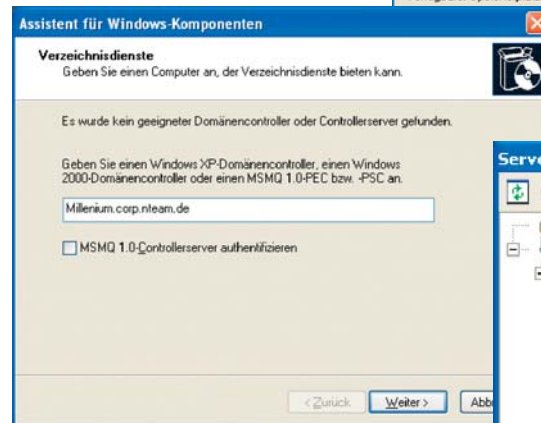


Abbildung 4 Auswahl des Domänen-Controllers für den Verzeichnisdienst.

kommt. Dies gilt unter Umständen auf alten Betriebssystemen. Als Client kommen folgende Windows Versionen in Frage:

- Win NT 4, Win 95, Win 98, Win Me (MSMQ 1.0),
- Win 2000 (MSMQ 2.0),
- Win XP (MSMQ 3.0).

Mit MSMQ 1.0 gab es einige Probleme, sodass der Einsatz von Windows 2000 oder Windows XP zu empfehlen ist. MSMQ läuft aber auch auf einem Windows-95-Rechner. Weitere Informationen finden Sie in [1], [2], [3] und [4].

### Warteschlangen

Unter MSMQ gibt es öffentliche (Public) und private (Private) Warteschlangen. Öffentliche Warteschlangen werden in das

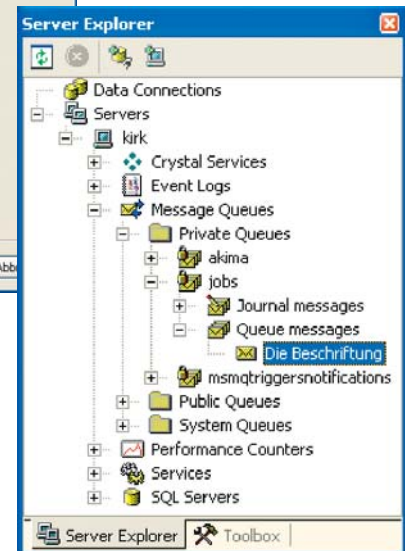


Abbildung 5 Der Server Explorer.

Active Directory eingetragen und können auch darüber gefunden werden. Wird MSMQ ohne Active Directory im Arbeitsgruppenmodus betrieben, dann können nur private Warteschlangen verwendet werden, deren genaue Adresse bekannt sein muss.

Beim Erstellen einer Warteschlange kann angegeben werden, ob Nachrichten dieser Warteschlange an Transaktionen teilnehmen sollen, sie kann als transaktional gekennzeichnet werden.

## Tabella 1

### Übersicht über die verschiedenen Adressierungsmöglichkeiten einer Queue.

FormatName:DIRECT=SPX:NetworkNumber;HostNumber\QueueName	Nur MSMQ 1,2
FormatName:DIRECT=TCP:IPAddress\QueueName	
FormatName:DIRECT=OS:MachineName\QueueName	
FormatName:DIRECT=OS:MachineName\Private\QueueName	
FormatName:DIRECT=HTTP://MSMQComputer/msmq/MyPublicQueue	Nur MSMQ 3
FormatName:DIRECT=HTTPS://MSMQComputer/msmq/MyPublicQueue	Nur MSMQ 3

Für die Verwaltung gibt es noch eine Reihe von Systemwarteschlangen, zum Beispiel für die Journalfunktion oder für unzustellbare Nachrichten, siehe Teil 2.

### Adressierung von Warteschlangen

Eine Warteschlange kann über folgende Eigenschaften adressiert werden: *Path*, *Label* und *FormatName*, im Workgroup-Modus steht nur *FormatName* zur Verfügung. Die ersten beiden Adressierungen werden mit Hilfe des MQIS in die reale Adresse (*FormatName*) der Warteschlange übersetzt. Der Zugriff über *FormatName* ist der schnellste, sicherste und für den Offline-Zugriff der einzige Weg, da offline kein MQIS verfügbar ist.

*Path* oder *Label* empfehlen sich nur, wenn sich die reale Adresse der Warteschlange häufig ändern kann, vielleicht, weil eine Applikation auf einen anderen Rechner im Active Directory umzieht. Werden Nachrichten ohne Zugriff auf den MQIS versandt, dann stehen auch einige Sicherheitsmechanismen von MSMQ nicht zur Verfügung (siehe Platform SDK:

Message Queuing). In Tabelle 1 finden Sie die verschiedenen Möglichkeiten für *FormatName*.

### System.Messaging und das Versenden einer Nachricht

Die Programmierung unter VB.NET und C# ist identisch. Als Erstes müssen Sie in einem neuen Projekt eine Referenz auf die *System.Messaging.dll* setzen. Danach verwenden Sie *Imports System.Messaging* um den Namensraum in Ihren Codemodulen bekannt zu machen.

Für die Arbeit mit MSMQ benötigen Sie zwei wichtige Klassen: *MessageQueue* und *Message*. Die erste Klasse ermöglicht das Lesen aus und das Schreiben in eine Queue, die zweite wird für die Nachrichten verwendet, die verschickt werden. Listing 1 zeigt das Öffnen einer Queue und versenden einer einfachen Nachricht, die aus einer Beschriftung (*Label*) und einem Text besteht.

Achtung, die *MessageQueue.Exists*-Methode kann nicht mit *FormatName*-Pfadangaben verwendet werden und der Aufruf ist zeitintensiv. Über die *Send*-Methode wird eine einfache Textnachricht versendet. Anstatt der Pfadangaben aus Listing 1 können Sie auch einen *FormatName* aus Tabelle 1 verwenden, um eine Queue zu öffnen

Ob der Versand ein Erfolg war, können Sie leicht prüfen. Dafür steht Ihnen die Computerverwaltung oder der Server Explorer [Strg]+[Alt]+[S] (Abbildung 5) von Visual Studio zur Verfügung. Im Server Explorer können Sie zu den Queues Ihres Computers navigieren und die Nachrichten einsehen. Markieren Sie eine Nachricht und drücken Sie dann [F4].

Dadurch erscheint das Property-Fenster und Sie können die verschiedenen Eigenschaften des *Message*-Objekts untersuchen. Mit dem Server Explorer und mit der Computerverwaltung können Sie sich auch zu anderen Computern verbinden und deren Warteschlangen einsehen. Bei

einer Workgroup-Installation geht das nur mit der Computerverwaltung.

### Das Lesen von Nachrichten

Für das Lesen von Nachrichten haben Sie mehrere Möglichkeiten. Die *Receive*-Methode entnimmt die Nachrichten aus der Warteschlange, die *Peek*-Methode schaut sich die Nachrichten nur an und lässt sie in der Warteschlange. Beiden Methoden kann ein Zeitraum (*TimeSpan*) übergeben werden, der festlegt, wie lange auf das Eintreffen einer Nachricht gewartet werden soll, falls sich keine Nachricht in der Queue befindet.

Beide Methoden blockieren so lange, bis entweder eine Nachricht gefunden wurde oder der Timeout auftritt. In Listing 2 finden Sie ein Beispiel für die *Receive*-Methode. Ihr wird ein *TimeSpan*-Objekt von zwei Sekunden übergeben. Kann in den zwei Sekunden keine Nachricht empfangen werden, dann wird eine

### Listing 2

#### Empfangen einer Nachricht.

```
Private Sub MSMQReceive()  
    Dim mq As MessageQueue  
    Dim msg As Message  
    'make sure that the Queue exists!  
    mq = New MessageQueue(kirk\Private$\Jobs)  
    Try  
        'waits 2 seconds for arrival of a message  
        msg = mq.Receive(New TimeSpan(0, 0, 2))  
        lstReceive.Items.Add(msg.Label)  
    Catch ex As Exception  
        MsgBox("No Message in the Queue!")  
    End Try  
End Sub
```

### Listing 3

#### Listen aller Nachrichten einer Schlange mittels Enumerator.

```
Private Sub MSMQShow()  
    Dim mq As MessageQueue  
    Dim msg As Message  
  
    'make sure that the Queue exists!  
    mq = New MessageQueue(m_QueuePath)  
    lstShow.Items.Clear()  
    For Each msg In mq  
        lstShow.Items.Add(msg.Label)  
    Next  
End Sub
```

### Listing 1

#### Öffnen einer Queue über eine Pfadangabe.

```
Private Sub MSMQSend()  
    Dim mq As MessageQueue, s As String  
    'Kirk is the local Computername  
    'Path Example for a Public Queue  
    'only if Active Directory is available  
    s = "Kirk\Jobs"  
    'Path Example for Private Queue  
    s = "kirk\Private$\Jobs"  
    If MessageQueue.Exists(s) Then  
        mq = New MessageQueue(s)  
    Else  
        mq = MessageQueue.Create(s)  
    End If  
    mq.Send("Hello Job", "The Label")  
End Sub
```

### Listing 4

#### Beispiel für asynchronen Empfang, eine Queue auf Modulebene und eine Ereignisbehandlungsroutine.

```
Public Class Form1
    Private WithEvents m_TheXQueue As Messaging.MessageQueue
    'Event from the Queue is fired after a Message could Received
    Private Sub m_TheXQueue_ReceiveCompleted(ByVal sender As Object, ByVal e As _
        System.Messaging.ReceiveCompletedEventArgs) Handles m_TheXQueue.ReceiveCompleted
        lstWatch.Items.Add(e.Message.Label)
        'restarts the asynchronous Receive
        m_TheXQueue.BeginReceive()
    End Sub

    'Starting the asynchronous Receive Process
    Private Sub cmdWatch_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles cmdWatch.Click
        m_TheXQueue = New MessageQueue(txtBigBrother.Text)
        m_TheXQueue.SynchronizingObject = Me 'for Multithreading issues
        m_TheXQueue.BeginReceive()
    End Sub
End Class
```

Exception ausgelöst, die abgefangen werden muss.

#### Einsehen von Nachrichten

Wenn Sie die Nachrichten einer Queue einsehen möchten, ohne sie zu entnehmen, können Sie beispielsweise *For Each* benutzen. Im .NET Framework spielen verschiedene Dinge nahtlos ineinander. Eine *MessageQueue* implementiert das Interface *IEnumerable*, was bedeutet, dass Sie mit der *For-Each*-Schleife über alle Nachrichten iterieren können. So können Sie sich den Inhalt einer Queue ansehen, ohne ihn zu verändern. Listing 3 zeigt ein Beispiel.

Ein *Message*-Objekt verfügt über eine Fülle von Eigenschaften, die Sie für Ihre Anwendung verwenden können. Alternativ zu *For Each* können Sie auch die *GetAllMessages*-Methode verwenden, die Ihnen als Schnappschuss ein Array mit allen Nachrichten der Warteschlange liefert.

#### Automatisches Empfangen von Nachrichten

Der große Vorteil von MSMQ-Anwendungen liegt darin, den Anwender über Änderungen sofort zu informieren. Das kann auf die klassische Art erreicht werden, indem ein Timer alle paar Millisekunden das Vorhandensein neuer Nachrichten prüft. Aber wer will schon immer pollen? Schöner wäre doch ein System, das von sich aus meldet, wenn neue Informationen eingetroffen

sind. Auch für diesen Fall bietet MSMQ einen Mechanismus, siehe Listing 4. Die Klasse *MessageQueue* verfügt über die beiden Events *ReceiveComplete* und *Peek-*

*Complete*. Sie müssen eine Nachrichtenwarteschlange mit dem Schlüsselwort *WithEvents* auf Modulebene deklarieren und eine entsprechende Ereignisbehandlungsprozedur schreiben, in der Sie auf das Eintreffen neuer Nachrichten reagieren. Listing 4 zeigt ein solches Beispiel.

Mit *BeginReceive* wird der Lesevorgang auf einem anderen Thread gestartet. Dadurch wird der Hauptthread nicht blockiert und die Anwendung kann ungestört weiterarbeiten. Trifft dann eine Nachricht ein, ruft der Receive-Thread die Ereignisprozedur auf. Damit das Zusammenspiel der beiden Threads reibungslos funktioniert, ist es wichtig, dass Sie die *SynchronizingObject*-Property auf die Instanz der Klasse setzen, die die Queue-Variable auf Modulebene enthält, in diesem Fall auf die Instanz der Formklasse. Achten Sie darauf, dass Sie nach dem Erhalt einer Nachricht den asynchronen Empfangsmechanismus erneut starten müssen.

#### Was kann versendet werden

Nur selten sollen einfache Textnachrichten versendet werden. In Listing 1 wurde

### Listing 5

#### Das Auspacken versendeter Objekte.

```
'Event from the Queue is fired after a Message was Received
Private Sub m_TheXQueue_ReceiveCompleted(ByVal sender As Object, ByVal e As _
    System.Messaging.ReceiveCompletedEventArgs) Handles m_TheXQueue.ReceiveCompleted
    Dim MessageFormatter As IMessageFormatter

    If e.Message.BodyType = 0 Then
        'XML Format
        Dim MsgTargetTypes(1) As Type
        MsgTargetTypes(0) = GetType(String)
        MsgTargetTypes(1) = GetType(CCustomer)
        MessageFormatter = New XmlMessageFormatter(MsgTargetTypes)
    Else
        'Binary Format
        MessageFormatter = New BinaryMessageFormatter
    End If

    e.Message.Formatter = MessageFormatter

    If TypeOf (e.Message.Body) Is CCustomer Then
        Dim theCustomer As CCustomer
        theCustomer = CType(e.Message.Body, CCustomer)
        lstWatch.Items.Add(theCustomer.Caption)
    ElseIf TypeOf (e.Message.Body) Is String Then
        lstWatch.Items.Add(e.Message.Label + " - " + CStr(e.Message.Body))
    Else
        lstWatch.Items.Add(e.Message.Label + " - unknown Bodytype!")
    End If

    'restarts the asynchronous Receive
    m_TheXQueue.BeginReceive()
End Sub
```

## Die Tücken des XMLMessageFormatter.

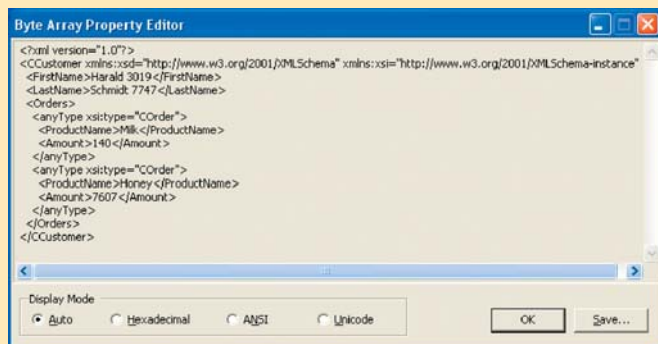
Achtung, wenn Sie den XML-Formatter verwenden und Objekte versenden. Dann müssen alle Klassen, die serialisiert werden sollen, über einen Default-Konstruktor (ein Konstruktor ohne Parameter) verfügen! Wenn Sie aus der Klasse *COrders* des Beispielprojektes den leeren Default-Konstruktor löschen, erhalten Sie zur Laufzeit einen XML-Fehler; fangen Sie die *InnerException* ab, dort finden Sie den Grund.

Der *XmlMessageFormatter* benötigt ein Schema für die Serialisierung. Wird ihm zur Laufzeit ein Objekt übergeben, dann versucht er ein Schema zu erstellen und entsprechend zu serialisieren. Dabei hat der Formatter ein Problem mit untypisierten Collection-Klassen wie der *ArrayList* und der *HashTable*. Er erkennt nicht, was für Elemente er dort finden wird. Deshalb ist es wichtig ihm mitzuteilen, was ihn erwartet.

Die Klasse *CCustomer* des Beispielprojektes auf der Heft-CD ist mit dem Attribut *Serializable* gekennzeichnet. Dem *BinaryMessageFormatter* reicht das völlig aus, der *XmlMessageFormatter* stolpert aber bei der Serialisierung der *Orders*-Eigenschaft über die *COrder*-Objekte und löst eine Exception aus. Deshalb müssen Sie über Klassen, die serialisiert werden sollen und solche untypisierten Unterelemente enthalten, das *XMLInclude*-Attribut schreiben und alle zu erwartenden Typen angeben.

```
<Serializable(), XmlInclude(GetType(COrder))> Public Class CCustomer
```

Der XML-Formatter ermöglicht das Lesen der Nachrichten (siehe Abbildung 6) und mag sinnvoll sein, wenn Nicht-.NET- Systeme mit der Nachricht etwas anfangen wollen, aber er ist langsamer und der Codierungsaufwand für das Ein- und Auspacken ist größer. Sie sollten nach Möglichkeit den Binär-Formatter einsetzen. Achtung, der XML-Formatter ist der Default-Formatter.



**Abbildung 6**  
Wenn Sie im Server Explorer eine Nachricht markieren, [F4] drücken und dann die Body-Stream-Property auswählen, sehen Sie den Inhalt der Nachricht.

eine Nachricht verschickt, die als Body und als Label einen String enthielt. Dem Body einer Nachricht kann auch eine Objektreferenz übergeben werden, das heißt, Sie können beliebige Objekte übertragen, die serialisierbar sein müssen (siehe den Kasten „Objektpersistenz“).

Angenommen, Sie haben ein Kundenobjekt mit einer Collection von Bestellobjekten und wollen das Kundenobjekt mit seinen Bestellungen versenden. Dazu wird der Body-Property des Message-Objektes die Referenz auf das Kundenobjekt übergeben, MSMQ verpackt dann automatisch im Hintergrund alle Objekte in einen Byte-Strom und versendet diesen über das Netzwerk.

In Listing 5 finden Sie eine Prozedur aus dem Beispielprojekt, die die versendeten Nachrichteninhalte auspackt. Beim Auspacken muss unterschieden werden, mit welchem Formatter die Nachricht eingepackt wurde (siehe den Kasten „Objektpersistenz“). Im Beispiel wird dafür die *BodyType*-Property des *Message*-Objektes verwendet. Alternativ dazu können Sie auch die *AppSpecific*-Property des *Message*-Objektes verwenden. Eine Applikation sollte nach Möglichkeit nur eine Art von Formatter einsetzen.

Dem *XMLMessageFormatter* müssen dann die zu erwartenden Typen mitgeteilt werden, ähnlich dem im Kasten „Objektpersistenz“ beschriebenen

*XMLInclude*-Attribut. Der binäre Formatter ist intelligenter und braucht diese Informationen nicht. Anschließend wird der erstellte Formatter dem empfangenen *Message*-Objekt übergeben und es kann geprüft werden, welcher Datentyp in der Nachricht steckt.

Mit dem Beispielprogramm können Sie das Versenden von Objekten testen. Achten Sie darauf, dass beim Objektversenden nicht mehr auf das Vorhandensein der Queue getestet wird. Wenn die Queue noch nicht existiert, dann versenden Sie als Erstes eine Textnachricht mit dem Knopf *Send one text Message* und prüfen den Versand mit dem Knopf *Show Content*. Dann entscheiden Sie sich für den XML-Formatter oder den binären Formatter und versenden die Objekte. Setzen Sie einen Breakpoint in der Ereignisprozedur und drücken Sie den Knopf *Receive asynchron*. Beobachten Sie das Auspacken der Objekte. Sie können auch in den Konstruktoren der Objekte Breakpoints setzen und können so auch das Instanzieren der Objekte sehen.

## Das Message-Objekt

In Listing 1 wurde eine Nachricht ohne ein *Message*-Objekt versendet. Wenn Sie ein *Message*-Objekt verwenden, dann können Sie eine Vielzahl von Optionen konfigurieren, die den Versand beeinflussen. Ohne *Message*-Objekt werden die *DefaultPropertiesToSend* des *Queue*-Objektes verwendet. Im zweiten Teil werde ich auf die verschiedenen Versendeoptionen des *Message*-Objektes eingehen.

## Fazit

Jetzt sollten Sie in der Lage sein, Experimente mit MSMQ durchzuführen. Der zweite Teil dieses Artikels wird sich mit den fortgeschrittenen Themen rund um MSMQ beschäftigen und MSMQ mit den konkurrierenden Technologien Remoting und Web Services vergleichen. |||||

- [1] Marcel Gnoth, Nachrichtenbasierte Informationssysteme, BasicPro 5/2001, Seite 26 ff.
- [2] MS-Newsgroup [microsoft.public.msmq.deployment](http://microsoft.public.msmq.deployment)
- [3] Installation of MSMQ - WC071602.ppt auf der Heft-CD
- [4] <http://support.microsoft.com/default.aspx?scid=kb;en-us;317329>
- [5] Marcel Gnoth, Konferenzen: [www.gnoth.net/BASTA/Feb02.htm](http://www.gnoth.net/BASTA/Feb02.htm)