# Kapitel 15

## Die ganze Mannschaft: Der StockQuote-Server

In Kapitel 8, »Das Projekt«, habe ich das Beispiel eines StockQuote-Servers verwendet, um die Vorteile der Komponenten-Architektur und die Entscheidungskriterien zwischen ActiveX-EXE-Servern, DLL-Servern und anderen Ansätzen darzustellen. Ich hatte dabei versprochen, Ihnen zu zeigen, wie man eine solche Anwendung erstellt.

In den vorangegangenen sechs Kapiteln haben wir uns mit den Hintergründen der Implementierung von ActiveX-Komponenten befaßt – in diesem Kapitel werden wir die frisch gewonnenen Kenntnisse zur Anwendung bringen.

## 15.1 Top-Down-Design

Es gibt verschiedene Entwurfsmethoden für Programme und ich könnte nicht sagen, welche die beste wäre. Ich bevorzuge selbst die Methode, den Entwurf von oben herab zu beginnen und den Code von unten her einzufügen. Dies bedeutet, daß Sie mit der grundlegenden Aufgabenstellung des Programms beginnen und dann auf dieser Ebene festlegen, welche Module, Komponenten und Funktionen zu implementieren sind. Dann gehen Sie eine Ebene tiefer und planen den Code für diese Module – und so weiter. Wenn es soweit ist, tatsächlich Code zu schreiben, arbeiten Sie in umgekehrter Richtung: Sie schreiben zunächst den Code der Module auf der untersten Ebene und arbeiten sich nach und nach aufwärts.

### 15.1.1 Die Grundstruktur des Entwurfs

Unser Ziel ist ein ActiveX-EXE-Server, der Aktien-Notierungen abfragen kann. Die Gründe für die Wahl eines EXE-Servers haben wir in Kapitel 8 besprochen. Diese waren im wesentlichen:

- Ein EXE-Server kann die Kurse in einem separaten Thread abfragen, so daß die abfragende Anwendung während des Abfrage-Vorgangs nicht blockiert wird.
- Ein EXE-Server, der mehrere Objekte offenlegt, erlaubt mehreren Client-Anwendungen den Zugriff auf die gleiche Kommunikationsverbindung.

Das heißt grundsätzlich, daß jede beliebige Anwendung eine Aktien-Notierung abfragen kann und eine Benachrichtigung erhält, wenn die Antwort eingetroffen ist. Eine Anwendung sollte auch mehrere Abfragen gleichzeitig anfordern können. Der Server muß mehrere Clients unterstützen.

Diese Anforderungen fließen in weitere Entwurfsentscheidungen ein. So stehen beispielsweise zwei verschiedene Benachrichtigungsmechanismen zur Verfügung: OLE-Ereignisse und OLE-Rückrufe. Welchen sollten wir wählen? Um diese Frage beantworten zu können, müssen Sie zuerst entscheiden, ob ein einzelnes `StockQuote`-Objekt mehrere Anfragen zugleich oder nur eine einzige bearbeiten können soll. Wenn es mehrere Anfragen bearbeiten soll, wäre der OLE-Ereignis-Mechanismus sinnvoll. In diesem Fall erscheint es jedoch angebrachter,

daß ein Client mehrere `StockQuote`-Objekte anlegen und damit mehrere Anfragen gleichzeitig starten kann. Daher erscheint der OLE-Rückruf-Mechanismus als die bessere Wahl.

Sollte es ein Multithread-Server sein? In diesem Fall ist Multithread nicht angebracht. Wir wollen, daß der Server die Anfragen der Clients der Reihe nach bearbeitet. Dies bedeutet, daß die Objekte miteinander kommunizieren können sollen, am besten über gemeinsam genutzte Datenstrukturen in einem globalen Modul. Das Apartment-Modell von Visual Basic unterbindet diese gemeinsame Nutzung. Dazu kommt, daß nur eine einzige Kommunikationsverbindung zur Verfügung steht, so daß Multithreading keinen Vorteil brächte. Eher würde wegen des Multithreading-Overheads bloß die Performance gedrückt. Wir brauchen also nur einen einzigen Thread.

Somit nimmt die Grundstruktur des Entwurfs Konturen an. Sie haben ein `StockQuote`-Objekt, das von jeder beliebigen Anwendung angelegt werden kann. Es soll über eine Methode verfügen, die zum Starten einer Kursabfrage aufgerufen wird. Dieser Methode wird eine Referenz auf ein Rückruf-Objekt zur Benachrichtigung übergeben. Ihr wird auch das Notierungssymbol des abzufragenden Kurses übergeben. Der Server soll in einem Modul die eingehenden Anfragen nachhalten. Sobald die gewünschten Daten eingetroffen sind, wird das Rückruf-Objekt zur Benachrichtigung der Client-Anwendung verwendet.

Wir können den folgenden Entwurf für das `StockQuote`-Objekt festhalten:

- `GetQuote (Symbol, callback)`: Eine Methode zum Starten des Abfrage-Vorgangs.
- `State`: Eine schreibgeschützte Eigenschaft, die den Abfrage-Status widerspiegelt.
- `Symbol`: Eine schreibgeschützte Eigenschaft, die das Kurssymbol enthält. Sie wird von der Methode `GetQuote` gesetzt.
- `LastPrice`: Eine schreibgeschützte Eigenschaft, die die letzte, aktuellste Notierung enthält.
- `PriorClose`: Eine schreibgeschützte Eigenschaft, die den letzten Schlußkurs enthält.
- `Change`: Eine schreibgeschützte Eigenschaft, die die Tagesänderung des Kurses enthält.
- `QuoteTime`: Eine schreibgeschützte Eigenschaft mit der Uhrzeit der Notierung.
- `High`: Eine schreibgeschützte Eigenschaft mit dem Tageshöchststand.
- `Low`: Eine schreibgeschützte Eigenschaft mit dem Tagesniedrigststand.

Wir müssen noch eine Rückruf-Methode definieren – die Methode, über die das Rückruf-Objekt verfügen muß, um die Benachrichtigung empfangen zu können:

- `QuoteUpdate(StockQuote)`: Eine Rückruf-Methode, die als Parameter eine Referenz auf das `StockQuote`-Objekt selbst zurückgibt.
- `CancelNotification`: Eine Methode, über die der Client die Benachrichtigungen stoppen kann – etwa wenn er abschließen will.

Sollen die oben genannten Kurs-Informationen in der Form von Strings vorliegen, wie sie empfangen werden, oder sollen sie in Währungs-Werte konvertiert werden? In diesem Objekt sind sie als Strings belassen, da Kursnotierungen traditionell als Integerzahl, gefolgt von einem Bruch, angegeben werden. Die Darstellung als String ist daher einfacher. Allerdings wären Hilfsfunktionen ganz praktisch, die den Wert ins Währungsformat konvertieren, damit numerische Vergleiche durchgeführt werden können. Es ist naheliegend, derartige Funktionen im `StockQuote`-Objekt unterzubringen:

- `QuoteToCurrency(quote As String)`: Eine Funktion, der die Kursnotierung als String übergeben wird und die den numerischen Wert als `Currency`-Datentyp zurückgibt.
- `CurrencyToQuote(quote As Currency)`: Eine Funktion, die den Kurs als Währungs-Wert übernimmt und ihn im `String`-Format mit Bruch-Angabe zurückgibt.

### 15.1.2 Die Entwurfsdetails

Sie werden bemerkt haben, daß wir ein kleines Detail bisher außen vor gelassen haben: Woher bekommen wir denn nun den Kurs selbst? Derartige Informationen bekommen wir natürlich aus der neuen Quelle aller Weisheit, dem Internet. o.k., mag sein, daß das Internet nicht die Quelle *aller* Weisheit darstellt, aber zumindest sind dort Aktiennotierungen hinterlegt, die Sie abfragen können, wenn Sie wissen, wie man eine Abfrage absetzt und eine HTML-Seite ausliest.

Es gibt mehrere Möglichkeiten, über Visual Basic auf das Internet zuzugreifen. Für diese Beispiel-Anwendung habe ich eine gewählt, die wahrscheinlich jedem Leser zur Verfügung steht, das Internet Transfer Control, das in den Professional- und Enterprise-Editionen von Visual Basic enthalten ist. Über dieses Control kann Ihr Visual-Basic-Programm auf das Internet zugreifen, wenn eine Internet-Verbindung über TCP/IP besteht. Wie Sie zu einem Internet-Zugang kommen und wie Sie Ihr System dazu konfigurieren, übersteigt den Inhalt dieses Buches. Sie werden im Buchhandel bestimmt eines finden, das Ihnen dabei hilft.

Wird werden gleich über die Implementierung sprechen. Die Entscheidung für das Internet-Control ist der letzte Baustein des Puzzles zur Definition der Architektur des Servers.

In Abbildung 15.1 sehen Sie den Gesamtentwurf des `StockQuote`-Servers. Wenn der Client einen Kurs abfragt, ruft das `StockQuote`-Objekt im allgemeinen das `QuoteEngine`-Modul auf. Das Modul fügt eine Referenz auf das anfragende

Objekt in eine Collection ein, die eine Warteschlange der anfragenden Objekte darstellt. Wenn die erste Anfrage eintrifft, ruft die `QuoteEngine` eine Methode eines Hilfsformulars auf, die daraufhin eine HTML-Anfrage über das Internet-Control absetzt. Ist die Übertragung abgeschlossen (oder wenn ein Fehler auftritt), löst das Control ein Ereignis aus. Das Control läßt damit die `QuoteEngine` wissen, daß ein Kurs eingetroffen ist, und übergibt die HTML-Information an das `StockQuote`-Objekt. Dieses parst (analysiert) den HTML-Code und extrahiert die Kursinformation, benachrichtigt den Client – und fertig!

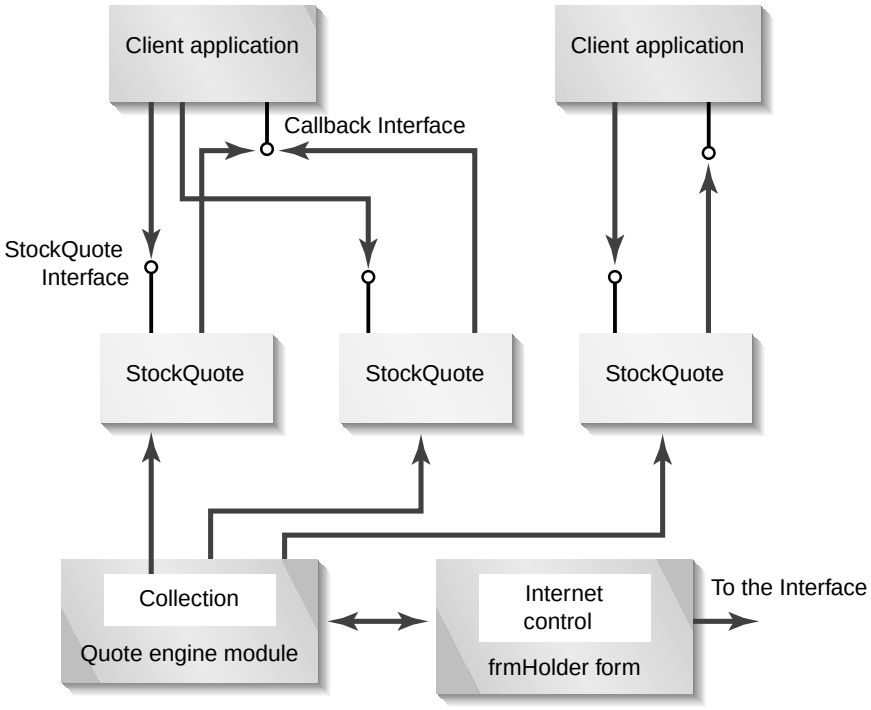


*Abb. 15.1: Architektur des StockQuote-Servers*

Das klingt eigentlich recht einfach. Und wenn Sie ein erfahrener Internet-Programmierer sind und derartiges bereits gemacht haben, ist es wirklich einfach. Dann können Sie sogar den Rest dieses Kapitels überspringen. Ich weiß aber, daß für viele unter Ihnen dies alles wie schwarze Magie vorkommt. Gehen wir also Schritt für Schritt vor, damit Sie sehen, wie das alles funktioniert.

## 15.2 Implementierung

Wir haben den Top-Down-Entwurf hinter uns. Nun steht die Codierung von unten herauf an.

### 15.2.1 Alles was Sie über HTML wissen müssen

Nun, nicht alles – aber alles, was Sie zum Erstellen dieser Anwendung wissen müssen.

Das `StockQuote`-Objekt ist hier so konfiguriert, daß es sowohl auf Notierungen von Charles Schwab (`www.schwab.com`) oder Yahoo (`www.yahoo.com`) zugreifen kann. Es gibt Myriaden weiterer Kurs-Dienste im Internet, für die Sie den Server zusätzlich konfigurieren können. Meine Wahl bedeutet keine Empfehlung oder Bevorzugung vor anderen. Ich gehe lediglich meistens über Yahoo ins Internet und ich habe ein persönliches Konto bei Schwab, so daß ich sie einfach netterweise in diesem Kontext erwähne. Aus meiner Perspektive kann ich nur sagen, daß es mich weniger als einen Tag kostete, von Schwab und Yahoo die Genehmigung für die Demonstration in diesem Projekt zu erhalten – was vielleicht ein gewisses (positives) Licht auf diese Unternehmen wirft.

Ich muß Sie allerdings auf eines aufmerksam machen. Nichts und niemand kann Yahoo und Schwab davon abhalten, das Format ihrer Webseiten jederzeit zu ändern – das ist übers Jahr bereits mehrmals vorgekommen. Es kann also durchaus passieren, daß diese Komponente nicht wie erwartet funktioniert und daß einige der Beschreibungen in diesem Abschnitt hinfällig werden. Sollte dies der Fall sein, werden Sie aktuelle Informationen auf der Web-Site meines Unternehmens unter `www.desaware.com` finden.

Damit Sie verstehen können, wie diese Internet-Abfrage verwirklicht wird, sollten Sie ein wenig über HTML und darüber wissen, wie das World Wide Web funktioniert.

Wenn Sie einem Browser (oder einem Internet-Control) mitgeteilt haben, daß Sie eine bestimmte Webseite zu erhalten wünschen, laufen verschiedene Dinge ab. Zunächst greift das Control auf einen Domain Name Service (DNS) zurück, um die eindeutige 64-Bit-Adresse des Computers zu erhalten, auf dem sich die gewünschte Seite befindet. Dies ist die Internet-Protocol-Adresse (IP-Adresse). Domänen sind hierarchisch angelegt. So ist beispielsweise die Site `www.desaware.com` Teil der `com`-Domäne, die kommerzielle Web-Sites umfaßt. Weiterhin ist die Site Teil einer Domäne namens `desaware`.

Nachdem das Control die IP-Adresse erhalten hat, öffnet es eine Kommunikationsverbindung zu der Site – diese wird »Socket« genannt. Auf der Site muß ein Server-Programm laufen, das auf eingehende Anfragen wartet. Dieses Programm wird »Web-Server« genannt.

Das Control sendet einen String mit einer Anfrage an die Site. Die Site antwortet mit den gewünschten Daten. Im http-Protokoll des World Wide Web können die zurückgegebenen Daten in Textform oder in binärer Form vorliegen. In unserem Fall benötigen wir die Textform.

Wenn Sie eine Seite von einer Web-Site abfragen, erhalten Sie einen Text-String in dem Format, das HyperText Markup Language (HMTL) genannt wird. Der String besteht aus Elementen zweierlei Typs:

- *Tags* – diese werden von spitzen Klammern eingeschlossen »<...>«; Tags enthalten Formatierungsinformationen, die Anweisungen für den Browser darstellen.
- *Daten* – der auf der Seite darzustellende Text bzw. Inhalt.

Wenn ein Browser eine recht komplexe Seite empfängt, die Bilder, Links und Animationen enthält, startet er eigentlich mehrere Anfragen nacheinander, im Hintergrund. Die Anfangsseite enthält Tags, die dem Browser mitteilen, wo er die übrigen Elemente finden kann, die zu der Seite gehören. Legt ein Tag fest, daß eine Abbildung geladen werden soll, muß der Browser eine weitere Anfrage absetzen, um das Bild von der im Tag angegebenen Quelle zu erhalten. Listing 15.1 zeigt, wie eine typische `Stockquote`-Seite von Schwab in HTML aussieht.

```
<TD ALIGN=LEFT>
<TABLE BORDER=0 WIDTH=155 CELLSPACING=0>
<TR BGCOLOR=#CCCCCC VALIGN=MIDDLE>
<TD ALIGN=LEFT>
<FONT FACE="Arial, Helvetica" SIZE=2 CLASS="bodyFont">
 Last Trade:</FONT></TD>
<TD ALIGN=RIGHT>
<FONT FACE="Arial, Helvetica" SIZE=2 CLASS="bodyFont">
54 15/16 </FONT></TD>
</TR>

<TR VALIGN=MIDDLE>
<TD ALIGN=LEFT>
<FONT FACE="Arial, Helvetica" SIZE=2 CLASS="bodyFont">
 Net Change:</FONT></TD>
<TD ALIGN=RIGHT>
<FONT FACE="Arial, Helvetica" SIZE=2 CLASS="bodyFont">
<FONT COLOR="#008800">+1 3/16</FONT> </FONT></TD>
</TR>
<TR BGCOLOR=#CCCCCC VALIGN=MIDDLE>
<TD ALIGN=LEFT>
<FONT FACE="Arial, Helvetica" SIZE=2 CLASS="bodyFont">
 Day High:</FONT></TD>
<TD ALIGN=RIGHT>
<FONT FACE="Arial, Helvetica" SIZE=2 CLASS="bodyFont">
55 </FONT></TD>
</TR>

<TR VALIGN=MIDDLE>
<TD ALIGN=LEFT>
```

```
<FONT FACE="Arial, Helvetica" SIZE=2 CLASS="bodyFont">
 Day Low:</FONT></TD>
<TD ALIGN=RIGHT>
<FONT FACE="Arial, Helvetica" SIZE=2 CLASS="bodyFont">
54 </FONT></TD>
</TR>

<TR BGCOLOR=#CCCCCC VALIGN=MIDDLE>
<TD ALIGN=LEFT>
<FONT FACE="Arial, Helvetica" SIZE=2 CLASS="bodyFont">
 Date:</FONT></TD>
<TD ALIGN=RIGHT>
<FONT FACE="Arial, Helvetica" SIZE=2 CLASS="bodyFont">
7/20/98 </FONT></TD>
</TR>

<TR VALIGN=MIDDLE>
<TD ALIGN=LEFT>
<FONT FACE="Arial, Helvetica" SIZE=2 CLASS="bodyFont">
 Trade Time:</FONT></TD>
<TD ALIGN=RIGHT>
<FONT FACE="Arial, Helvetica" SIZE=2 CLASS="bodyFont">
16:01 </FONT></TD>
</TR>

<FORM METHOD="POST" ACTION="perf_snapshot">
<INPUT TYPE="HIDDEN" NAME="color" VALUE="Blue, Green on White">
<INPUT TYPE="HIDDEN" NAME="size" VALUE="240 x 180 GIF">
<INPUT TYPE="HIDDEN" NAME="volume" VALUE="off">
<TD ALIGN=LEFT COLSPAN=2>
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0 WIDTH=290>
<TR>
<TD ALIGN=LEFT NOWRAP>
<FONT FACE="Arial, Helvetica" SIZE=2 CLASS="bodyFont">
<NOBR>New Performance Snapshot:</NOBR></FONT></TD>
<TD ALIGN=LEFT>
<FONT FACE="Arial, Helvetica" SIZE=2 CLASS="bodyFont">
<input name="symbols" type="text" size="6" VALUE="">
</FONT></TD><TD WIDTH=2%> </TD>
<TD ALIGN=LEFT>
<INPUT TYPE="IMAGE"
SRC="http://gsf.quote.com/fast/graphics/public2/submit.gif"
ALIGN="top" BORDER=0></TD>
</TR>
```

*Listing 15.1: Auszug aus der Quotation-HTML-Seite*

Wenn Sie genau hinschauen, werden Sie feststellen, daß bereits die meisten der Informationen, die wir benötigen, in diesem Listing enthalten sind. Sie sehen auch einige Beispiele von Tags. Beispielsweise sehen Sie hier ein Tag, das ein Bild abfragt, das angeklickt werden kann, um Informationen an ein Formular zu übergeben:

```
<INPUT TYPE="IMAGE"
SRC="http://gsf.quote.com/fast/graphics/public2/submit.gif"
ALIGN="top" BORDER=0></TD>
```

Das Bild ist nicht Bestandteil der Seite selbst. Ein Browser muß eine separate Anfrage absetzen, um das Bild von der angegebenen Quelle zu laden. Wir werden uns das Listing später noch einmal anschauen, wenn es darum geht, die gewünschten Informationen aus der Seite zu extrahieren.

Die Frage lautet nun, wie man den Kurs einer bestimmten Aktie erhalten kann. Dazu schauen wir uns einen Teil der Hauptseite der Aktiennotierungen an:

```
<html>

<head>

<SCRIPT LANGUAGE="JavaScript">
<!--
function snapShot()
{
if (location.search.length > 0)
{
var value =
location.search.substring(6, location.search.length);
main.location.href =
"http://fast.quote.com/fq/public2/perf_snapshot?symbols=" +
value +
"&size=240+x+180+GIF&color=Blue,+Green+on+White&volume=off";
}
}

//-->
</SCRIPT>


</head>

<frameset rows="76, *" frameborder="0" framespacing="0"
border="0" onLoad="snapShot();">
<frame src="/SchwabNOW/headers/CSHead.html" marginwidth="0"
marginheight="0" scrolling="no"
noresize name="head">
```

Diese Seite ist nicht gerade leicht zu verstehen. Das Problem liegt darin, daß die Hauptseite auf mehreren Frames (Rahmen) basiert. Die Seite, die uns interessiert, befindet sich im ersten Rahmen und hat einen Parameter namens `onLoad`, der die JavaScript-Funktion `snapShot()` aufruft. Die Schlüsselzeile dieser Funktion lautet:

```
main.location.href =
"http://fast.quote.com/fq/public2/perf_snapshot?symbols=" +
value
```

Diese Zeile enthält einen URL und eine Folge von Angaben zur Abfrage einer Notierungsseite. Wie können wir das prüfen? Geben Sie folgendes in die Adreßzeile Ihres Browsers ein:

```
http://fast.quote.com/fq/public2/perf_snapshot?symbols=msft
```

Sie brauchen kein HTML-Experte zu sein, um das zu verstehen. Sie brauchen bloß ein wenig Geduld und Experimentierfreude. Ich habe es beispielsweise herausgefunden, indem ich die Seiten besucht habe, an denen ich interessiert war, und mir den Quelltext angesehen (das geht in jedem besseren Browser) und den HTML analysiert. Allerdings kann das etwas vertrackt sein, wenn es sich um eine Seite mit Frames handelt. Im Microsoft Internet Explorer können Sie mit der rechten Maustaste den Rahmen anklicken, dessen Quelltext Sie sehen wollen, und dann im Kontextmenü den Punkt QUELLTEXT ANZEIGEN auswählen. Auch in Netscapes Navigator kann man sich den Quelltext aller Frames ansehen. Alternativ dazu können Sie auch den URL einer Frame-Teilseite in die Adreßzeile des Browsers eingeben und so die Seite für sich alleine im Browser-Fenster ansehen.

So erhalten wir die Informationen darüber, wie das Formular in der Notierungsseite aus Listing 15.1 funktioniert. Die folgenden drei Zeilen ermöglichen die Abfrage einer Notierung über ein Formular:

```
<FORM METHOD="POST" ACTION="perf_snapshot">
<input name="symbols" type="text" size="6" VALUE="">
<INPUT TYPE="IMAGE"
SRC="http://gsf.quote.com/fast/graphics/public2/submit.gif"
ALIGN="top" BORDER=0>
```

Wird die Submit-Grafik angeklickt, wird die Aktion des Formulars ausgeführt. Hier geht die Anfrage an das Objekt `perf_snapshot` im aktuellen Verzeichnis der Site, das, wie wir bereits gesehen haben, `http://fast.quote.com/fq/public2` lautet. Die TextBox heißt »symbols«.

Der Server erhält die übrigen Informationen in einer Befehlszeile, die er bearbeitet, um die Antwortseite zu erstellen, die die Aktiennotierung enthält. Die von einem Web-Formular zusammengestellte Befehlszeile besteht aus einem Fragezeichen, gefolgt von den Bezeichnungen der einzelnen Formular-Felder (wie der TextBox) und den Werten, die durch Ampersands (»&«) voneinander getrennt werden. Allgemein lautet das Format:

```
http://webaddress.domain/Action?firstfield=firstvalue
&secondfield=secondvalue&
```

Die Anfrage nach Microsofts Kursnotierung lautet somit:

```
http://fast.quote.com/fq/public2/perf_snapshot?symbols=msft
```

Sie sehen, Sie brauchen gar nicht zu wissen, was die verschiedenen HTML-Tags bedeuten, um Informationen aus dem Internet abrufen zu können. Ich weiß auch nicht, was diese Tags alle bedeuten. Wir werden uns in Teil II noch einmal um HTML kümmern, wo es um Web-Seiten geht, die ActiveX-Controls und -Dokumente enthalten. Doch nun sind wir soweit, uns wieder mit Code beschäftigen zu können.

### 15.2.2 »Talking to The Net«

Das Internet-Control ist eigentlich recht einfach zu verwenden. Sie setzen die Eigenschaft `Protocol` auf `4 - icHTTP`. Das Control befindet sich auf einem Formular namens `frmHolder`, einem unsichtbaren Formular des EXE-Servers.

Das Formular verfügt über die öffentliche Methode `StartQuote`, über die der Abfrage-Vorgang gestartet wird. Es folgt die simple Routine:

```
Public Sub StartQuote(symbol As String)
   Dim q$
   Select Case QuoteSource
      Case sqschwab
         q$ = _
          "http://fast.quote.com/fq/public2/" _
          & perf_snapshot?symbols=" & Trim$(symbol)
         Inet1.Execute q$, "GET"
      Case sqyahoo
         q$ = "http://quote.yahoo.com/q?s=" _
          & Trim$(symbol) & "&d=t"
         Inet1.Execute q$, "GET"
   End Select
End Sub
```

Sie sehen, wir machen hier nichts weiter, als den Abfrage-String zusammenzusetzen, den der Server für eine bestimmte Notierung erwartet. Dann rufen wir die `Execute`-Methode des Controls auf, um die Anfrage an den Server abzusetzen.

Das Control erledigt die ganze Arbeit, den Domänen-Namen aufzulösen und die Seite zu empfangen. Sie können den Vorgang über das `StateChanged`-Ereignis des Controls verfolgen (siehe Listing 15.2).

```
Private Sub Inet1_StateChanged(ByVal State As Integer)
  Dim res$
  Dim ChunkVar As Variant
```

```
Dim bDone As Boolean
Dim st$
Select Case State
   Case icNone
   Case icResolvingHost ' Das Control schaut nach
                        ' der IP Adresse des
                        ' angegebenen Host-Computers
      st$ = "Resolving host"
   Case icHostResolved ' Das Control hat die
                       ' IP-Adresse des angegebenen
                       ' Host-Computers gefunden
      st$ = "Host resolved"
   Case icConnecting ' Das Control stellt die
                  ' Verbindung zum Host-Computer her
      st$ = "Connecting"
   Case icConnected           ' Das Control hat die
                    ' Verbindung zum Host-Computer
                    ' hergestellt
      st$ = "Connected"
   Case icRequesting ' Das Control sendet
                     ' eine Anfrage an den
                     ' Host-Computer
      st$ = "Requesting"
   Case icRequestSent ' Das Control konnte die
                      ' Anfrage absenden
      st$ = "Request sent"
   Case icReceivingResponse ' Das Control erhält eine
                            ' Antwort vom Host-Computer
      st$ = "Receiving"
   Case icResponseReceived ' Das Control hat die Antwort vom
                           ' Host-Computer erhalten
      st$ = "Response received"
      DoEvents
   Case icDisconnecting ' Das Control trennt die
                        ' Verbindung zum Host-Computer
      st$ = "Disconnecting"
   Case icDisconnected ' Das Control hat die
               ' Verbindung zum Host-Computer getrennt
      st$ = "Disconnected"
   Case icError ' Ein Fehler ist in der Kommunikation
                ' mit dem Host-Computer aufgetreten
      st$ = "Error"
      EndQuote ""
   Case icResponseCompleted ' Die Anfrage ist
                            ' abgeschlossen und alle
                            ' Daten sind eingegangen
      st$ = "Response complete"
```

```
            Do
               DoEvents
               On Error Resume Next
               ChunkVar = Inet1.GetChunk(1024, icString)
            Loop While Len(ChunkVar) = 0
            On Error GoTo 0
            Do
               If Len(ChunkVar) > 0 Then
                  res = res & ChunkVar
                  DoEvents
                  ChunkVar = Inet1.GetChunk(1024, icString)
                  DoEvents
               Else
                  bDone = True
               End If
            Loop While Not bDone
            EndQuote res
      End Select
      Debug.Print st$
   End Sub

   Private Sub Form_Terminate()
      DoEvents ' Notwendig wegen eines MSInet-Bugs
   End Sub
```

*Listing 15.2: Die Ereignisse Inet1_StateChanged und Form_Terminate*

Wie Sie sehen, liefert das Control jederzeit Informationen darüber, was es gerade tut. In diesem Beispiel wird die Anweisung `Debug.Print` dazu verwendet, Ihnen anzuzeigen, was sich gerade tut.

Für diese Komponente sind lediglich zwei Zustände von Interesse. Der Status `icError` zeigt an, daß ein Fehler aufgetreten ist. Die am häufigsten auftretenden Fehler werden Timeout-Fehler bei Verbindungs- oder (Netzwerk-)Server-Problemen sein. Der Status `icResponseCompleted` informiert das Programm darüber, daß die Daten empfangen wurden. Die Methode `GetChunk` des Controls liefert einen String gegebener Länge. Wir durchlaufen solange eine Schleife, bis die Seite vollständig in einen String aufgenommen worden ist.

Die abschließende Operation bei beiden Zustandsmeldungen ist der Aufruf der `EndQuote`-Funktion, die Bestandteil des Moduls `QuoteEngine` ist. Dies teilt der Engine mit, daß eine Antwort eingetroffen ist.

Sie werden sich wundern, warum nach jedem `GetChunk`-Aufruf, nach jedem `ResponseRecieved`-Zustand und `Terminate`-Ereignis des Formulars jeweils eine `DoEvents`-Anweisung eingefügt ist – da ich doch von der Verwendung von `DoEvents` abrate. In den meisten Fällen ist die Verwendung von `DoEvents` ein

Anzeichen für einen mangelhaften Entwurf. Sie werden sich auch wundern, warum der Code so angelegt ist, Fehler zu ignorieren, die während der ersten `GetChunk`-Operation auftreten.

Hier dienen die `DoEvents`-Anweisungen zur Umgehung eines Bugs im Microsoft-Internet-Control. Sie lösen ein bekanntes Synchronisationsproblem des Controls. Der Bug ist in der Readme-Datei zur Hilfe von Visual Basic 5 dokumentiert. Er könnte in Visual Basic 6 beseitigt worden sein – doch dann sind wohl neue Bugs an dessen Stelle getreten: Auf einigen Systemen löst der erste `GetChunk`-Aufruf einen Fehler aus und gibt keine Daten zurück – ganz klar ein Bug.

Ich habe dieses Control in diesem Beispiel verwendet, da ich sicher sein kann, daß jeder Leser darüber verfügt. Aber Sie müßten, ganz locker gesagt, schon verrückt sein, das Control in einer echten Anwendung einzusetzen. Es ist einfach viel zu instabil und Microsoft hat sich nicht viel Mühe gegeben, die Rückwärtskompatibilität zu wahren – es kann vorkommen, daß einige Anwendungen mit neueren Versionen des Controls nicht mehr einwandfrei zusammenarbeiten. Wenn Sie einen Internet-Zugriff dieser Art realisieren wollen, sollten Sie sich den Gefallen tun und ein qualitativ besseres, kommerzielles Winsock-Control verwenden, zu dem es auch einen besseren Support gibt.

### 15.2.3 Das Modul QuoteEngine

Der Code, der die verschiedenen `StockQuote`-Objekte verwaltet und mit dem Netzwerk über das Formular `frmHolder` kommuniziert, steckt in einem Standard-Modul namens `QuoteEngine`. Aber da wir doch so sehr um objekt-orientierte Programmierung bemüht sind, wäre doch ein Standard-Modul das letzte, was wir in einer Komponente verwenden sollten, werden Sie jetzt sicher denken. In der Regel ist das auch richtig. Jedoch stellt das `QuoteEngine`-Modul aus der Sicht der Komponente selbst eine Art Objekt dar, auch wenn es kein COM-Objekt ist. Da die Variablen eines Standard-Moduls global für die gesamte Anwendung sein können, stellt es das ideale Mittel zur gemeinsamen Nutzung von Informationen durch die Objekte der Anwendung und zur Vermittlung zwischen diesen dar. Die `QuoteEngine` ist ein exzellentes Beispiel dafür, wann und wie man ein Standard-Modul einsetzen sollte.

In Listing 15.3 sehen Sie den Code des `QuoteEngine`-Moduls. Dort werden mehrere Techniken verwendet, die Sie bereits früher gesehen haben. So wird in der `Sub Main`-Prozedur ermittelt, ob die Komponente als Stand-alone-Programm gestartet wurde. Ist dies der Fall, wird ein zweites Formular `frmQuote.frm` geöffnet. Dieses Formular enthält eine TextBox, in der Sie die Anfrage nach verschiedenen Kursen eingeben können. Zugleich ist dies eine einfache Möglichkeit, die Komponente zu testen, ohne eine separate zweite Anwendung zu benötigen. Es ist nicht unangebracht, diese Technik zum Testen von ActiveX-EXE-Servern ganz allgemein zu verwenden. Sie können dieses Formular schließlich auch vor der Auslieferung der Komponente entfernen oder so sperren, daß es nie angezeigt

wird. Wurde der Server als Komponente gestartet, setzen wir die Eigenschaft `TaskVisible` des `App`-Objekts auf `False`, so daß die Komponente nicht in der Task-Liste des Systems auftaucht.

```
     ' dwQuote QuoteEngine
' Copyright (c) 1997-1998 by Desaware Inc.  All Rights Reserved

     Option Explicit

     ' Enumeration zur Festlegung des gewünschten Dienstes
     Public Enum QuoteSourceType
        sqschwab = 0   ' Charles Schwab & Co.
        sqyahoo = 1    ' Yahoo
     End Enum

     ' verwendeter Dienst
     Public QuoteSource As QuoteSourceType

     ' Dies ist eine Collection der Kurse, die gesucht werden
     ' sollen. Sie werden der Reihe nach ermittelt. Alle
     ' Objekte arbeiten mit dieser Auflistung.
     Private QuotesPending As New Collection

     ' Gleich True, wenn gerade eine Abfrage stattfindet
     Private QuotationInProgress As Boolean

     ' Initialisierungs-Routine
     Sub Main()

        ' Nur in diesem Beispiel hart-codierte Dienst-Wahl
        QuoteSource = sqyahoo

        ' Beim Stand-Alone-Lauf Abfrage-Formular öffnen
        If App.StartMode = vbSModeStandalone Then
           frmQuote.Show
        Else
           ' Anwendung nicht in der Task-Liste anzeigen
           App.TaskVisible = False
        End If
     End Sub

     ' Wird vom StockQuote-Objekt zum Start einer Abfrage
     ' aufgerufen, wenn nicht bereits eine gerade aktiv ist
     Public Sub StartQuote(Optional obj As StockQuote)
        Dim sq As StockQuote
        ' In Liste aufnehmen
        If Not obj Is Nothing Then
           QuotesPending.Add obj
```

```
    End If
    If (Not QuotationInProgress) And _
     QuotesPending.Count > 0 Then
       QuotationInProgress = True
       Set sq = QuotesPending.item(1)
       ' Abfrage jetzt starten:
       frmHolder.StartQuote sq.symbol
    End If
End Sub

' Wird aufgerufen, wenn eine Abfrage abgeschlossen ist
' htmlstring ist ein String, der die heruntergeladene
' HTML-Seite enthält
' Er ist leer, wenn ein Fehler auftritt
Public Sub EndQuote(htmlstring As String)
    Dim htmlcol As New dwHTMLcollection
    Dim sq As StockQuote

    ' Abfrage abgeschlossen
    QuotationInProgress = False

    ' Erstes Element aus der Collection entfernen
    ' (dasjenige das gerade abgearbeitet worden ist)
    Set sq = QuotesPending(1)
    QuotesPending.Remove 1

    ' Unbenötigte Objekte nicht herumliegen lassen
    If QuotesPending.Count = 0 Then
       Set QuotesPending = Nothing
       Unload frmHolder
    End If

    If Len(htmlstring) = 0 Then
       sq.ReportQuote Nothing, sqError
    Else
       htmlcol.LoadFromString htmlstring
       sq.ReportQuote htmlcol, sqIdle
    End If
    ' Nächste Abfrage starten
    StartQuote
End Sub
```

*Listing 15.3: Das Modul QuoteEngine*

Das Modul hat zwei private Variablen. Privat heißt hier, daß auf sie nur aus Funktionen dieses Moduls heraus zugegriffen werden kann. Daß das Standard-Modul von allen anderen Modulen der Anwendung gemeinsam genutzt wird, ist noch

lange kein Grund dafür, nicht auch hier objekt-orientierte Techniken anzuwenden, um diejenigen Variablen und Funktionen zu verbergen, die nur innerhalb des Moduls verwendet werden, eher im Gegenteil. Die Collection `QuotePending` enthält Referenzen auf alle `StockQuote`-Objekte, deren Anfragen zur Bearbeitung anstehen. Die boolesche Variable `QuotationInProgress` hält fest, ob eine Abfrage gerade bearbeitet wird.

Warum hält eine Standard-Collection die zur Bearbeitung anstehenden `StockQuote`-Objekte? Warum legen wir keine eigene Collection-Klasse oder kein Array an? Weil das hier reine Verschwendung an Zeit und Mühe und Code wäre. Der Hauptgrund für das Anlegen eigener Collections ist robusterer Code und die Minimierung der Gefahr, daß Clients der Collection ungültige Objekte einfügen oder eine unbeabsichtigte Manipulation der Collection vornehmen können. Doch hier ist der einzige Client der Collection das Modul selbst. Es sollte daher weitaus einfacher sein, sicherzustellen, daß der Code des Moduls korrekt ist, als eine eigene Collection anzulegen und auszutesten.

Das zusätzliche Objekt in einer eigenen Collection ist zudem eine Verschwendung von Code und Ressourcen. Der geringe Geschwindigkeitsvorteil, den die Verwendung eines Arrays hier bieten würde, ist auch vernachlässigbar im Verhältnis zu der langen Bearbeitungszeit jeder einzelnen Anfrage.

Über die öffentliche Variable `QuoteSource` können Sie den Kurs-Dienst festlegen. Hier ist das nur eine vorübergehende Implementierung zu Testzwecken. Sie sollte vielleicht bei einer endgültigen Implementierung dieser Komponente in das `StockQuote`-Objekt verlegt werden.

Das `QuoteEngine`-Modul hat nur zwei öffentliche Funktionen. Die Funktion `StartQuote` wird von einem `StockQuote`-Objekt zur Mitteilung aufgerufen, daß eine Abfrage erfolgen soll. Das `StockQuote`-Objekt übergibt als Parameter eine Referenz auf sich selbst (»`Me`«). Die Objekt-Referenz wird dann der `QuotesPending`-Collection hinzugefügt. Ist eine Abfrage gerade in Bearbeitung, kehrt die Funktion zurück. Anderenfalls wird geprüft, ob bereits andere Anfragen zur Bearbeitung anstehen. Ist dies der Fall, wird diese gestartet.

Die Funktion erfüllt daher zweierlei Aufgaben. Übergeben Sie ihr nämlich im Parameter als Referenz Nothing, startet die Funktion die nächste anstehende Abfrage, wenn eine vorhanden ist. So kann sie auch von der Funktion `EndQuote` aufgerufen werden, um die nächstfolgende Abfrage zu starten. Die Funktion `Endquote` wird nur vom Formular `frmHolder` aus aufgerufen, wenn eine Abfrage abgeschlossen ist.

Die Funktion `EndQuote` entfernt zuerst dasjenige `StockQuote`-Objekt aus der `QuotesPending`-Collection, das die aktuelle Abfrage angestoßen hatte, hält aber eine Referenz darauf fest, damit es nicht gelöscht wird. Stehen keine weiteren Abfragen an, wird das Formular entladen. Dies ist notwendig, damit der Server ordentlich beendet werden kann (ein EXE-Server kann erst dann beendet werden, wenn alle seine Formulare entladen sind).

Die Funktion `EndQuote` ruft dann die Methode `ReportQuote` desjenigen `StockQuote`-Objekts auf, das die Abfrage angestoßen hatte. Dem Aufruf werden zwei Parameter übergeben, ein `dwHTMLcollection`-Objekt, das im nächsten Abschnitt beschrieben wird, und ein Flag, das angibt, ob die Anfrage erfolgreich abgeschlossen wurde oder ob ein Fehler aufgetreten ist. Das Objekt `dwHTMLcollection` wurde über die Methode `LoadFromString` mit der empfangenen Webseite initialisiert.

Schließlich startet die Funktion `EndQuote` die nächstfolgende Abfrage. Sie sehen, dies ist eine recht elegante Lösung zur Abfrage von Kurs-Notierungen. Der EXE-Server läuft in seinem eigenen Thread – er läuft also im Hintergrund der Clients, die ihn verwenden. Die eigentliche Datenabfrage über das Internet läuft auch in bezug auf den Server im Hintergrund, wobei der Server die ganze Zeit über geöffnet bleibt, um weitere `StockQuote`-Objekte anzubieten und mit Abfragen zu füllen.

### 15.2.4 HTML parsen

Sie haben gesehen, daß HTML-Seiten aus einem String aus Tag-Daten und aus Inhaltsdaten bestehen. Wie extrahieren wir die Informationen aus der Seite, die wir benötigen? Der Vorgang des Extrahierens von Informationen aus einem String wird »Parsen« genannt – und genau das werden wir hier tun.

Wenn dies hier das einzige Mal ist, daß Sie eine HTML-Seite parsen müssen, dann können Sie auch die `Instr$`-Funktion zur Suche und zum Extrahieren von Teilstrings verwenden. Doch ich bin davon ausgegangen, daß ich diese Funktionalität häufiger brauchen kann, so daß ich mich um eine allgemeinere Lösung bemüht habe.

Als erstes müssen wir die Seite speichern. Eine Möglichkeit besteht darin, sie weiterhin als String zu erhalten, vielleicht auch als ein HTML-Seiten-Objekt mit Methoden zum Extrahieren von Informationen aus der Seite. Ein anderer Ansatz wäre, die Seite in mehrere Objekte zu zerlegen, wobei jedes Objekt einen einzelnen Tag- oder Inhaltsstring repräsentiert. Dieser zweite Ansatz ist weitaus flexibler. Denn wenn Sie einmal eine Seite in ihre Elemente zerlegt haben, ist es sehr einfach, neue Elemente hinzuzufügen oder die Elemente neu anzuordnen. Das ist besonders beim Erstellen von Server-Scripts von Vorteil. Ebenso geht auch die Suche nach bestimmten Elementen recht flott von der Hand. Der Nachteil besteht darin, daß das Anlegen eines jeweils neuen Objekts für jedes Element der Seite relativ ineffizient ist.

Die hier gezeigte Implementierung beruht auf dem zweiten Ansatz. Die Seite wird in ihre einzelnen Bestandteile zerlegt, die jeweils in einem Objekt namens `dwHTMLelement` abgelegt werden. Dieses Objekt sehen Sie in Listing 15.4.

```
' HTML element
 Desaware ActiveX Gallimaufry
' Copyright (c) 1997 by Desaware Inc.  All Rights Reserved
```

```
Option Explicit

Private intTag As String
Private intContents As String
Private separator As String

' Tag. Leerer String für Text
Public Function Tag() As String
   Tag = intTag
End Function

' Inhalt des Tag-Elements oder Text (wenn kein Tag)
Public Function Contents() As String
   Contents = intContents
End Function

' Lädt das nächste Tag- oder String-Element
Public Function LoadFromString(inputstring As String) _
 As String
   Dim Bracket1Pos As Long
   Dim Bracket2pos As Long
   Dim holdstring As String
   Dim spacepos As Long
   Dim returnstring As String
   Bracket1Pos = InStr(inputstring, "<")
   Bracket2pos = InStr(inputstring, ">")
   If Bracket1Pos > 1 Then
      ' Alles bis zur ersten Klammer ist Inhalt für
      ' dieses Objekt
      holdstring = Left$(inputstring, Bracket1Pos - 1)
      returnstring = Mid$(inputstring, Bracket1Pos)
   Else
      If Bracket1Pos = 0 Then
         ' Kein Tag vorhanden. Der übergebene String ist
         ' durchgängig Inhalt dieses Objekts
         holdstring = inputstring
      Else
         ' Anfang des Tags
         If Bracket2pos = 0 Then ' keine rechte Klammer!
            ' Dies sollte auf einer gültigen Seite
            ' niemals vorkommen
            ' Wird als String behandelt
            returnstring = inputstring
         Else
            ' Tag für dieses Objekt vollständig erhalten
            holdstring = LTrim$(Mid$(inputstring, _
             2, Bracket2pos - 2))
```

```
            If Bracket2pos < Len(inputstring) Then
               returnstring = _
               Mid$(inputstring, Bracket2pos + 1)
            End If
         End If
      End If
   End If
   If holdstring <> "" Then
      If Bracket1Pos <> 1 Then ' Kein Tag - nur Inhalt
                               ' speichern
         intContents = holdstring
      Else
         ' Es ist ein Tag.
         ' Erstes Trennzeichen finden
         spacepos = StringSpan(holdstring, separator$)
         If spacepos <= 1 Then
            ' Kein Trennzeichen - daher Tag ohne
            ' Parameter
            intTag = holdstring
         Else
            ' Tag im Element intTag speichern,
            ' die Parameter im Element intContents
            intTag = Left$(holdstring, spacepos - 1)
            If spacepos < Len(holdstring) Then
               intContents = Mid$(holdstring, _
               spacepos + 1)
            End If
         End If
      End If
   End If
   ' Links abschneiden
   spacepos = StringSpan2(returnstring, separator)
   If spacepos = 1 Then
      LoadFromString = returnstring
   Else
      If spacepos > 0 Then
         LoadFromString = Mid$(returnstring, spacepos)
      End If
   End If
End Function

Private Sub Class_Initialize()
   ' Trennzeichen-String initialisieren
   separator = " " & vbCrLf & vbTab
End Sub
```

*Listing 15.4: Die Klasse dwHTMLelement*

Jedes Objekt enthält zwei wesentliche Datenelemente. Die Tag-Daten enthalten den Typ des Tags, wenn das Objekt ein Tag repräsentiert. Die Inhaltsdaten enthalten die Textdaten sowohl von Nicht-Tag-Elementen oder von Parametern eines Tag-Elements. Beide Elemente werden über Funktionen offengelegt.

Das Objekt wird über die Funktion `LoadFromString` geladen, der der HTML-Code als Parameter-String übergeben wird. Diese Methode lädt das Objekt aus dem ersten Objekt des Strings. Dann gibt es den HTML-String um das erste Element gekürzt zurück. Sie werden gleich sehen, wie dies zum schnellen Parsen einer ganzen Seite eingesetzt wird. Das Objekt verwendet zwei Hilfsfunktionen zur String-Bearbeitung, die in einem separaten Modul definiert sind (siehe Listing 15.5).

```
' String functions
' Desaware ActiveX Gallimaufry
' Copyright (c) 1997 by Desaware Inc.  All Rights Reserved

Option Explicit

' Ursprungs-String sourcestring nach erstem Auftreten
' eines in searchchars enthaltenen Zeichens durchsuchen
Public Function StringSpan( _
 sourcestring As String, searchchars As String) As Long
   Dim x&
   Dim strlen&
   Dim foundpos&
   For x = 1 To Len(sourcestring)
      If InStr(searchchars, Mid$(sourcestring, x, 1)) _
       > 0 Then
         StringSpan = x
         Exit Function
      End If
   Next
End Function

' Ursprungs-String sourcestring nach erstem Auftreten
' eines nicht in searchchars enthaltenen Zeichens
' durchsuchen.
' Rückgabewert 0, wenn String komplett in searchchars
Public Function StringSpan2(sourcestring As String, _
 searchchars As String) As Long
   Dim x&
   Dim strlen&
   Dim foundpos&
   For x = 1 To Len(sourcestring)
      If InStr(searchchars, Mid$(sourcestring, x, 1)) _
       = 0 Then
         StringSpan2 = x
```

```
         Exit Function
      End If
   Next
End Function

' Literale HTML-Zeichen in reale Zeichen umwandeln
Public Function ConvertHtmlLiterals( _
 strInput As String) As String
   Dim amppos&
   Dim semipos&
   Dim lit$
   Dim res$
   Dim charval As Integer
   amppos = InStr(strInput, "&")
   If amppos = 0 Then
      ' Keine Literale, einfach kopieren
      ConvertHtmlLiterals = strInput
      Exit Function
   End If
   semipos = InStr(strInput, ";")
   If semipos = 0 Or semipos <= amppos + 1 Then
      ' kein HTML-Literal-String, einfach kopieren
      ConvertHtmlLiterals = strInput
      Exit Function
   End If
   ' String holen
   lit$ = Mid$(strInput, amppos + 1, semipos - amppos _
    - 1)
   If Left$(lit$, 1) = "#" Then
      ' Numerisches Literal
      charval = Val(Mid$(lit$, 2))
   Else
      Select Case LCase$(lit$)
         Case "lt"
            charval = 60
         Case "gt"
            charval = 62
         Case "amp"
            charval = 38
         Case "quot"
            charval = 34
         Case "emdash"
            charval = 151
         Case "copy"
            charval = 169
         Case "reg"
            charval = 174
```

```
            Case Else
               charval = 0
         End Select
      End If
      ' Unbekanntes Symbol heraustrennen
      If amppos > 1 Then
         res = Left$(strInput, amppos - 1)
      End If
      If charval <> 0 Then
         res = res & Chr$(charval)
      End If
      If semipos < Len(strInput) Then
         res = res & Mid$(strInput, semipos + 1)
      End If

      ' Warum Rekursion? Es könnte mehr als ein Literal
      ' enthalten sein
      ConvertHtmlLiterals = ConvertHtmlLiterals(res)
   End Function

   ' Linefeeds entfernen
   Public Function StripLinefeeds(line$) As String
      Dim x%
      x% = InStr(line$, Chr$(10))
      If x = 0 Then StripLinefeeds = line _
       Else StripLinefeeds = Left$(line$, x - 1)
End Function
```

*Listing 15.5: String-Funktionen im Modul strFuncs.bas*

Die Funktion `StringSpan` sucht nach dem ersten Auftreten eines Zeichens in einem übergebenen String, das einem Zeichen in einem Such-String entspricht. Die Funktion `StringSpan2` hält es umgekehrt, indem sie nach dem ersten Zeichen sucht, das *nicht* in einem Such-String enthalten ist. Diese Funktionen dienen zum Finden von Trennzeichen zwischen Elementen.

Das Modul `strFuncs` enthält ebenfalls die Funktion `ConvertHtmlLiterals`. Da in HTML verschiedene Symbole zum Verknüpfen und Formatieren von Seiten verwendet werden, ist es notwendig zu unterscheiden, wo diese Symbole als Steueranweisungen verwendet werden und wo sie im Text auftreten. Wenn beispielsweise die Zeile  im Text einer Seite erscheint, sollte der Browser diese als Center-Tag interpretieren. In HTML wird dazu ein Weg definiert, spezielle Zeichen zu kennzeichnen: Einem kaufmännischen Und (Ampersand-Zeichen – »&«) folgen ein Ascii- oder ein Text-Code und ein Semikolon. Um nun die Zeichenfolge  im Text selbst verwenden zu können, müßten Sie folgendes eingeben:

```
&lt;center&gt;
```

Die Funktion `ConvertHtmlLiterals` durchsucht einen String und konvertiert alle speziellen HTML-Codes in die tatsächlichen Zeichen, damit sie in Ihrem Programm einfach verwendet werden können.

Die Funktion `StripLineFeeds` liefert den Teil vor dem ersten Auftreten eines Linefeed-Zeichens (Zeilentrennung) in einem String. Dies wird später zum Extrahieren von Informationen aus einer Seite benötigt.

### 15.2.5 HTML-Elemente sammeln

Sobald eine HTML-Seite mit Kursnotierungen eingetroffen ist, müssen zwei Dinge erledigt werden. Zunächst muß die Seite in eine Collection von `dwHTMLelement`-Objekten zerlegt werden. Dann muß die Collection nach den gesuchten Informationen durchsucht werden.

Da es sich anbietet, diese Elemente-Collection nicht nur mit dem `StockQuote`-Objekt zu verwenden, sondern auch in anderen Komponenten, ist sie damit ein geeigneter Kandidat für eine selbsterstellte Collection. Die Verwendung einer eigenen Collection hat über den offensichtlichen Sicherheitsaspekt hinaus noch zwei weitere Vorteile. Sie ermöglicht einen effizienteren Ansatz auf Array-Basis zur Speicherung der Objekt-Referenzen, und in ihr können ein paar zusätzliche, für `dwHTMLelement`-Objekte spezifische Suchfunktionen untergebracht werden.

In Listing 15.6 sehen Sie die Collection-bezogenen Methoden und Eigenschaften dieser Klasse. Die hier verwendete Array-basierte Technik ist weitgehendst identisch zu der in Kapitel 12 gezeigten. Die interessanteste Funktion ist hier die Funktion `LoadFromString`, die eine komplette HTML-Seite einliest und zerlegt und dabei die Collection mit neu angelegten `dwHTMLelement`-Objekten füllt.

```
' HTML element collection
' Copyright (c) 1997-1998 by Desaware Inc.  All Rights Reserved

    Option Explicit

    ' Lokale Variable für Collection
    Private mCol() As dwHTMLelement
    Private mColUsed As Long   ' Anzahl belegter Elemente
    Private mColSize As Long   ' Anzahl Elemente insgesamt
    Private Const GRANULARITY = 5

    Public Function Append(objNewMember As dwHTMLelement)
       If mColSize = mColUsed Then
          ' Array muß vergrößert werden
          mColSize = mColSize + GRANULARITY
          ReDim Preserve mCol(mColSize)
       End If
```

```
   mColUsed = mColUsed + 1
   Set mCol(mColUsed) = objNewMember
End Function

' Anzahl der Objekte in der Collection auslesen
Public Property Get Count() As Long
   Count = mColUsed
End Property

Public Property Get item(vntIndex As Long) _
 As dwHTMLelement
   If vntIndex < 1 Or vntIndex > mColUsed Then
      RaiseError 9   ' Index-Fehler
   End If
   Set item = mCol(vntIndex)
End Property

' Angegebenes Element entfernen
Public Sub Remove(vntIndex As Variant)
   Dim ctr&
   If vntIndex < 1 Or vntIndex > mColUsed Then
      RaiseError 9   ' Index-Fehler
   End If
   ' Inhalt des Arrays verschieben
   For ctr = vntIndex To mColUsed - 1
      Set mCol(ctr) = mCol(ctr + 1)
   Next ctr
   Set mCol(mColUsed) = Nothing
   mColUsed = mColUsed - 1

   If mColSize - mColUsed > GRANULARITY * 2 Then
      mColSize = mColSize - GRANULARITY
      ReDim Preserve mCol(mColSize)
   End If
End Sub

Private Sub Class_Initialize()
   ' Raum für erstes Element bereitstellen
   ReDim mCol(GRANULARITY)
   mColSize = GRANULARITY
End Sub

Private Sub Class_Terminate()
   ' mCol-Array-Objekte terminieren hier
End Sub

' Komplette Seite in Collection von
```

```
' dwHTMLelement-Objekten einlesen
Public Function LoadFromString(ByVal inputstring _
 As String)
   Dim CurrentElement As dwHTMLelement
   Do
      Set CurrentElement = New dwHTMLelement
      inputstring _
       = CurrentElement.LoadFromString(inputstring)
      Append CurrentElement
   Loop While Len(inputstring) > 0
End Function
```

*Listing 15.6: Die Methoden und Eigenschaften der Collection dwHTMLcollection*

In Listing 15.7 sehen Sie die Sucherweiterungen der Collection, mit denen bestimmter Inhalt der Seite ausfindig gemacht werden kann.

```
' Element mit einem bestimmten Tag-Inhaltspaar finden.
' Leerer Inhalt trifft immer zu.
' Berücksichtigung von Groß-/Kleinschreibung
' ist Standard
' Rückgabewert 0, wenn nichts gefunden wird.
Public Function Find(FirstElement As Long, ByVal Tag _
 As String, Optional ByVal Contents As String, _
 Optional CaseSensitive = True) As Long
   Dim LastElement&
   Dim CurElement&
   Dim thtml As dwHTMLelement
   Dim bFoundTag As Boolean
   Dim bFoundContents As Boolean
   LastElement = Count()
   ' Nicht damit aufhalten, wenn Ende erreicht
   If FirstElement > Count Then Exit Function
   ' Großschreibung prüfen, wenn Case-insensitiv
   If Not CaseSensitive Then
      Tag = UCase$(Tag)
      Contents = UCase$(Contents)
   End If
   For CurElement = FirstElement To LastElement
      Set thtml = mCol(CurElement)
      ' Zuerst Tag prüfen
      If Tag <> "" Then
         If Not CaseSensitive Then
            If Tag = UCase$(thtml.Tag) Then
               bFoundTag = True
            End If
         Else
            If Tag = thtml.Tag Then
```

```
                bFoundTag = True
             End If
          End If
       Else
          bFoundTag = True
       End If
       ' Nun Inhalt prüfen
       If Contents <> "" Then
          If Not CaseSensitive Then
             If Contents _
              = StripLinefeeds(UCase$(thtml.Contents)) _
              Then
                bFoundContents = True
             End If
          Else
             If Contents _
              = StripLinefeeds(thtml.Contents) Then
                bFoundContents = True
             End If
          End If
       Else
          bFoundContents = True
       End If
       ' Treffer, wenn beides zutrifft
       If bFoundTag And bFoundContents Then
          Find = CurElement
          Exit Function
       End If
    Next

End Function

' Sucht nach einer Folge von Tags
' ab Element FirstElement.
' CaseSensitive legt Berücksichtigung von
' Groß-Kleinschreibung fest
' Dann folgt eine Liste der zu findenden Tags.
' Die Tags müssen in der angegebenen Reihenfolge
' erscheinen
Public Function FindTagSequence(FirstElement As Long, _
 CaseSensitive As Boolean, ParamArray TagSequence() As _
 Variant) As Long
   Dim CurrentBase As Long
   Dim LastElement As Long
   Dim CurrentParam As Long
   Dim FirstParam As Long
   Dim LastParam As Long
```

```
   Dim bCompareFailed As Boolean
   ' Erwartete und aktuelle Tag-Position
   Dim ExpectedPosition As Long
   Dim CurrentPosition As Long

   LastElement = Count()
   CurrentBase = FirstElement
   LastParam = UBound(TagSequence)
   FirstParam = LBound(TagSequence)
   Do
      bCompareFailed = False
      ' Erste Übereinstimmung probieren
      CurrentBase = _
       Find(CurrentBase, TagSequence(FirstParam), , _
       CaseSensitive)
      ' Wenn keine Übereinstimmung, dann hier verlassen
      If CurrentBase = 0 Then Exit Function
      ' Nun die weiteren Parameter prüfen
      ExpectedPosition = CurrentBase + 1
      For CurrentParam = FirstParam + 1 To LastParam
         CurrentPosition = Find(ExpectedPosition, _
         TagSequence(CurrentParam), , CaseSensitive)
         ' Falsche Position, also hier verlassen
         If ExpectedPosition <> CurrentPosition Then
            bCompareFailed = True
            Exit For
         End If
         ' Erwarteten Wert erhöhen
         ExpectedPosition = ExpectedPosition + 1
      Next CurrentParam
      If Not bCompareFailed Then
         ' Vergleich für alle Tags erfolgreich!
         FindTagSequence = CurrentBase
         Exit Function
      End If
      CurrentBase = CurrentBase + 1
   Loop While CurrentBase <= LastElement

End Function

' Nächstes Element ohne Tag finden
' Suche endet bei CloseTag, wenn vorhanden
Public Function FindNextNonTag(Optional FirstLoc As _
 Long = 1, Optional CloseTag As String) As Long
   Dim LastElement&
   Dim CurElement&
   Dim thtml As dwHTMLelement
```

```
   LastElement = Count()
   For CurElement = FirstLoc To LastElement
      Set thtml = mCol(CurElement)
      If thtml.Tag = "" Then
         FindNextNonTag = CurElement
         Exit Function
      Else
         If thtml.Tag = CloseTag Then Exit For
      End If
   Next
End Function

' FindNextContent beginnt bei FirstLoc,
' Sucht nach String "FirstContent" in einem Inhaltsfeld
' - CaseSensitive!
' Sucht dann nach dem nächsten Nicht-Tag-Feld und gibt
' dieses zurück
' Aktualisiert den Parameter FirstLoc entsprechend der
' Position des zurückgegebenen Tags
' Gibt leeren String bei erfolgloser Suche zurück.
' Diese Funktion ist hilfreich zur Suche von
' Inhaltspaaren
Public Function FindNextContent(FirstLoc As Long, _
 FirstContent As String, Optional CaseSensitive _
 As Boolean = True) As String
   Dim HtmlFoundIdx&
   HtmlFoundIdx = FirstLoc
   HtmlFoundIdx = Find(HtmlFoundIdx, "", FirstContent, _
    CaseSensitive)
   If HtmlFoundIdx > 0 Then HtmlFoundIdx _
    = FindNextNonTag(HtmlFoundIdx + 1)
   If HtmlFoundIdx > 0 Then
      FindNextContent = mCol(HtmlFoundIdx).Contents
      FirstLoc = HtmlFoundIdx
   End If

End Function

' AppendThroughTag  beginnt mit FirstLoc,
' Sucht nach String "FirstContent" in einem Inhaltsfeld
' Hängt dann alle Nicht-Tag-Felder bis zum schließenden
' Tag an
' Aktualisiert den Parameter FirstLoc entsprechend der
' Position des zurückgegebenen Tags
' Gibt leeren String bei erfolgloser Suche zurück.
' Diese Funktion ist hilfreich zum Zusammenfügen von
```

```
' Inhaltsfeldern, die durch Formatierungs-Tags getrennt
' sind.
Public Function AppendThroughTag(FirstLoc As Long, _
 FirstContent As String, ByVal CloseTag As String) _
 As String
   Dim HtmlFoundIdx&
   Dim BuildString$
   HtmlFoundIdx = FirstLoc
   CloseTag = UCase$(CloseTag)
   If FirstContent <> "" Then HtmlFoundIdx _
    = Find(HtmlFoundIdx, "", FirstContent)
   If HtmlFoundIdx > 0 Then HtmlFoundIdx _
    = FindNextNonTag(HtmlFoundIdx + 1, CloseTag)
   If HtmlFoundIdx > 0 Then
      Do While HtmlFoundIdx <= Me.Count
         If UCase$(mCol(HtmlFoundIdx).Tag) = CloseTag _
          Then Exit Do
         If mCol(HtmlFoundIdx).Tag = "" Then
            BuildString = BuildString & _
             StripLinefeeds(mCol(HtmlFoundIdx).Contents)
         End If
         HtmlFoundIdx = HtmlFoundIdx + 1
      Loop
      FirstLoc = HtmlFoundIdx
   End If
   AppendThroughTag = BuildString
End Function

' Löst internen Fehler aus
Private Sub RaiseError(ByVal errnum&)
   Dim ErrNumToUse&
   If errnum >= 1000 Then
      ' Eigene Fehlernummer
      Err.Raise vbObjectError +             errnum, _
       "dwHTMLcollection", GetErrorString(errnum), _
       App.HelpFile, errnum + HelpBaseOffset
   Else
      ' Visual Basic-Fehler auslösen
      Err.Raise errnum, "dwHTMLcollection", _
       GetErrorString(errnum)
   End If
End Sub
```

*Listing 15.7: Suchfunktionen der Collection dwHTMLcollection*

In diesem Listing sehen Sie die folgenden Methoden:

Die `Find`-Funktion sucht nach einem bestimmten Element, sowohl anhand des Tags als auch des Inhalts. Sie können angeben, ob Groß-/Kleinschreibung berücksichtigt werden soll. Wird als Tag ein leerer String übergeben, sucht die Funktion nur nach dem angegebenen Inhalt. Ist der Inhalts-String leer oder fehlt er, sucht die Funktion nur nach dem angegebenen Tag. Wird beides angegeben, muß beides übereinstimmen. Die Funktion beginnt ab einer angegebenen Position und gibt die Position der ersten Übereinstimmung zurück, oder sie gibt 0 zurück, wenn keine Übereinstimmung gefunden wurde.

Die Funktion `FindTagSequence` sucht nach einer bestimmten Folge von Tag-Elementen. Wir brauchen ja eine Möglichkeit, genau die gewünschten Teile der HTML-Seite ausfindig zu machen. Da Seiten-Formate relativ konsistent sind, stellt eine Tag-Folge eine gute Möglichkeit dar, den Suchbereich zu begrenzen. Diese Funktion demonstriert auch ganz gut die Verwendung von Parameter-Arrays, da eine Funktion so mit beliebig vielen Parametern arbeiten kann.

Die Funktion `FindNextNonTag` wird dann gebraucht, wenn Sie die Tag-Folge gefunden haben, die den gewünschten Seiten-Bereich begrenzt. Diese Funktion überspringt alle verbleibenden Tags der Folge, um zur Position des ersten Nicht-Tag-Elements der Seite zurückzukehren. Sie können auch ein Ende-Tag angeben, das die Suche beendet. Das ist besonders dann sinnvoll, wenn das gesuchte Element leer sein sollte.

Die Funktion `FindNextContent` ist dann sehr hilfreich, wenn HTML-Text wie der folgende bearbeitet werden soll:

```
<td width=130>Today's open</td><td>155 1/8</td>
```

Die Beschreibung des Felds ist ein Inhaltselement, daß durch eines oder mehrere Tags von dem eigentlich gewünschten getrennt wird. Die Funktion kann beispielsweise den String »Today's open« als Parameter übernehmen, diesen Text im HTML-String finden, alle Tag-Elemente bis zum nächsten Inhalts-Element überspringen und den Inhalt dieses Elements zurückgeben – alles in einer einzigen Operation. Diese Funktion werden wir am häufigsten zum Extrahieren der Kurs-Informationen aus der Seite benötigen.

Die Funktion `AppendThroughTag` ist hilfreich bei der Behandlung von HTML-Code wie folgendem:

```
<TH ALIGN=right>Day Low:</TH><TD>94 <SUP>5</SUP>/<SUB>8</SUB></TD></TR>
```

Sie sucht nach einem angegebenen Inhalts-Tag und fügt dann alle Inhalts-Tags bis zu einem angegebenen Ende-Tag an. Hier könnte als Anfangs-Tag »Day Low« und als Ende-Tag »/TD« angegeben werden, um den String »94 5/8« zu erhalten.

Die Klasse enthält dazu eine zentrale Fehlerbehandlung. Das Modul `INetErrors` (hier nicht zu sehen) enthält die Fehlerkonstanten für die Komponente und die Funktion `GetErrorString`.

### 15.2.6 ... und nun das StockQuote-Objekt

Listing 15.8 enthält schließlich das eigentliche `StockQuote`-Objekt. Der größte Teil des Listings dürfte entsprechend Ihrem derzeitigen Kenntnisstand ziemlich klar sein. Die verschiedenen privaten Variablen nehmen die Kurz-Informationen auf. Die Variable `m_Notify` hält die Referenz auf das Rückruf-Objekt. Die Komponente bietet eine öffentliche Enumeration, damit die Anwendung den Wert der `State`-Funktion leichter interpretieren kann. Schreibgeschützte liefern die Kursnotierungen.

```
' Copyright (c) 1997-1998 by Desaware Inc.  All Rights Reserved

    Option Explicit

    ' Das Symbol wird über die GetQuote-Funktion gesetzt
    Private m_Symbol As String

    ' Dies wird von der Web-Seite geladen
    Private m_CompanyName As String
    Private m_LastPrice As String
    Private m_PriorClose As String
    Private m_Change As String
    Private m_High As String
    Private m_Low As String
    ' Zeit der letzten Notierung
    Private m_QuoteTime As String

    ' Beim Eintreffen zu benachrichtigendes Objekt.
    ' Objekt muß über folgende Funktion verfügen:
    ' Sub QuoteUpdate (QuoteInfo As StockQuote)
    ' Wir verwenden Rückrufe anstelle von Ereignissen, da
    ' die Anwendung in der Regel mehrere dieser Objekte hat.
    Private m_Notify As Object

    ' 0 - unbeschäftigt
    ' 1 - beschäftigt
    ' 2 - Fehler beim letzten Kurs
    Private m_State As Integer

    Public Enum QuoteState
       sqIdle = 0
       sqBusy = 1
       sqError = 2
    End Enum
```

```
' Status des Objekts erfragen
' 0 - unbeschäftigt
' 1 - beschäftigt
' 2 - Fehler bei letzter Operation
Public Property Get State() As QuoteState
   State = m_State
End Property

' Symbol
Public Property Get symbol() As String
   symbol = m_Symbol
End Property

' Eigenschaften zu Kurs-Information
Public Property Get CompanyName() As String
   CompanyName = m_CompanyName
End Property

Public Property Get LastPrice() As String
   LastPrice = m_LastPrice
End Property

Public Property Get PriorClose() As String
   PriorClose = m_PriorClose
End Property

Public Property Get Change() As String
   Change = m_Change
End Property

Public Property Get QuoteTime() As String
   QuoteTime = m_QuoteTime
End Property

Public Property Get High() As String
   High = m_High
End Property

Public Property Get Low() As String
   Low = m_Low
End Property

' Verhindert Benachrichtigung, wenn die Anwendung
' geschlossen werden soll. Zwingt das Quote-Objekt zur
' Freigabe des Rückruf-Objekts.
Public Sub CancelNotification()
   Set m_Notify = Nothing
End Sub
```

```
' Startet den Vorgang der Kurs-Abfrage
Public Sub GetQuote(symbol As String, _
 Optional callback As Object)
   If m_State = 1 Then
      ' Keine Abfrage, wenn bereits eine läuft
      TriggerError 4000
      Exit Sub
   End If
   If Not IsMissing(callback) Then
      Set m_Notify = callback
   End If
   ' Symbol setzen
   m_Symbol = symbol
   ' Als beschäftigt markieren
   m_State = 1
   ' Abfrage starten
   StartQuote Me
End Sub

' Fehler auslösen
Private Sub TriggerError(errnum As Long)
   Err.Raise vbObjectError + errnum, "StockQuote", _
    GetErrorString(errnum), App.HelpFile, errnum _
    + HelpBaseOffset
End Sub

' Kurs an das Rückruf-Objekt zurückgeben, wenn eines
' existiert
' Beachten Sie, daß dies ein Fehler ist - der
' Aufrufer sollte immer auf Fehler prüfen.
' HTML ist eine Collection von dwHTMLelements
Friend Sub ReportQuote(html As dwHTMLcollection, _
 endState As QuoteState)
   Dim CallbackObject As Object

   ' HTML-Seite parsen
   If Not html Is Nothing Then
      ParseHtml html
   End If

   ' Aktuellen Status setzen
   m_State = endState

   Set CallbackObject = m_Notify
   ' Rückruf immer löschen, wir halten ihn nicht
   Set m_Notify = Nothing
   If Not CallbackObject Is Nothing Then
```

```
            On Error GoTo NoCallbackName
            CallbackObject.QuoteUpdate Me
         End If
      NoCallbackName:
         ' Wenn ein Textfehler beim Rückruf erfolgt, verlassen
      End Sub

      Friend Sub ParseHtml(html As dwHTMLcollection)
         Select Case QuoteSource
            Case sqschwab
               ParseSchwabQuote html
            Case sqyahoo
               ParseYahooQuote html
         End Select
      End Sub

Private Sub ParseYahooQuote(html As dwHTMLcollection)
   Dim HtmlFoundIdx As Long
   Dim HtmlFoundIdx2 As Long
   Dim IsFund As Boolean
   Dim QuoteType$
   HtmlFoundIdx = html.FindTagSequence(1, False, "Table", _
    "tr", "td", "b")
   If HtmlFoundIdx = 0 Then Exit Sub
   HtmlFoundIdx2 = html.FindTagSequence(HtmlFoundIdx + 1, _
    False, "Table", "tr", "td", "b")
   If HtmlFoundIdx2<>0 then HtmlFoundIdx = HtmlFoundIdx2
          ' Erste Tabelle überspringen

   m_CompanyName = ConvertHtmlLiterals(html.item( _
    HtmlFoundIdx + 4).Contents)
   m_QuoteTime _
    = ConvertHtmlLiterals(html.FindNextContent(_
     HtmlFoundIdx, "Last Trade"))
   If m_QuoteTime = "" Then
      m_QuoteTime = ConvertHtmlLiterals( _
       html.FindNextContent(HtmlFoundIdx, "Net Asset Value"))
      IsFund = True
   End If
   HtmlFoundIdx = HtmlFoundIdx + 1
   m_LastPrice = ConvertHtmlLiterals(_
    html.AppendThroughTag(HtmlFoundIdx, "", "/b"))
   m_Change = html.AppendThroughTag(HtmlFoundIdx, "Change", _
    "/TD")
   m_PriorClose = ConvertHtmlLiterals( _
    html.AppendThroughTag(HtmlFoundIdx, "Prev Cls", "/td"))
```

```
   ' Wenn QuoteType <> "Stock" dann IsFund = True
End Sub

Private Sub ParseSchwabQuote(html As dwHTMLcollection)
   Dim HtmlFoundIdx As Long
   Dim IsFund As Boolean
   Dim QuoteType$
   'HtmlFoundIdx = html.FindTagSequence(1, False, "/Table", _
    "P")
   HtmlFoundIdx = 1
   HtmlFoundIdx = html.FindTagSequence(HtmlFoundIdx, False, _
    "TD", "NORB", "FONT", "B")

   If HtmlFoundIdx = 0 Then
      HtmlFoundIdx = html.FindTagSequence(1, False, "TD", _
       "FONT", "B")
      m_CompanyName = ConvertHtmlLiterals( _
       html.item(HtmlFoundIdx + 3).Contents)
   Else
      m_CompanyName = ConvertHtmlLiterals( _
       html.item(HtmlFoundIdx + 4).Contents)
   End If

   m_LastPrice = ConvertHtmlLiterals( _
    html.FindNextContent(HtmlFoundIdx, " Last Trade:"))
   If m_LastPrice = "" Then
      m_LastPrice = ConvertHtmlLiterals( _
       html.FindNextContent(HtmlFoundIdx, " NAV:"))
      IsFund = True
   End If
   m_Change = ConvertHtmlLiterals( _
    html.FindNextContent(HtmlFoundIdx, " Net Change:"))
   m_High = ConvertHtmlLiterals( _
    html.FindNextContent(HtmlFoundIdx, " Day High:"))
   m_Low = ConvertHtmlLiterals( _
    html.FindNextContent(HtmlFoundIdx, " Day Low:"))
   m_QuoteTime = ConvertHtmlLiterals( _
    html.FindNextContent(HtmlFoundIdx, " Date:"))
   m_QuoteTime = m_QuoteTime & " " & _
    ConvertHtmlLiterals(html.FindNextContent(HtmlFoundIdx, _
    " Trade Time:"))

End Sub

   Public Function QuoteToCurrency(ByVal quote As String) _
    As Currency
         Dim spacepos%   ' Position von Leerzeichen
```

```
    Dim fractionpos% ' Position des Bruchs
    Dim TempResult As Currency
    If quote = "" Then Exit Function
    quote = Trim$(quote) ' Führende und folgende
                         ' Leerzeichen abschneiden
    spacepos = InStr(quote, " ")
    If spacepos = 0 Then
        QuoteToCurrency = CCur(quote)
        Exit Function
    End If
    ' Integer-Wert zuerst
    TempResult = CCur(Left$(quote, spacepos - 1))

    ' Wir wissen nach der Trim-Funktion, daß da kein
    ' führendes Leerzeichen sein kann
    quote = Mid$(quote, spacepos + 1)

    ' Ist hier ein Bruch?
    fractionpos = InStr(quote, "/")
    If fractionpos <= 1 Or fractionpos = Len(quote) Then
        ' Wir wissen nicht, wie das zu parsen ist
        QuoteToCurrency = TempResult
        Exit Function
    End If
    On Error GoTo MathError:
    TempResult = TempResult + CCur(Left$(quote, _
     fractionpos - 1)) / CCur(Mid$(quote, fractionpos _
     + 1))
    QuoteToCurrency = TempResult
    Exit Function

MathError:
    ' Erstmal vorhandenen Wert zurückgeben
    QuoteToCurrency = TempResult
End Function

Public Function CurrencyToQuote(ByVal quote _
 As Currency) As String
    Dim IntVal As Integer
    Dim FracVal As Integer
    Dim Denominator As Integer
    IntVal = Fix(quote) ' Integer-Anteil holen

    ' Bruch-Anteil holen
    ' Wie viele 64stel  (kleinste gewünschte Teilung)
    FracVal = CInt(Abs(quote - IntVal) * 64)
```

```
        ' Dividieren solange gerade
        Denominator = 64
        Do
            If (FracVal And 1) Then
                ' Ist ungerade
                Exit Do
            End If
            FracVal = FracVal \ 2
            Denominator = Denominator \ 2
        Loop While Denominator > 0

        CurrencyToQuote = IntVal & " " & FracVal & "/" _
         & Denominator

    End Function
```

*Listing 15.8: Das Listing des StockQuote-Objekts*

Die wahrscheinlich interessantesten Funktionen sind hier `ParseHtml` und die sie aufrufenden Funktionen. Diese Funktionen schauen zuerst nach den Tags, die vor dem Unternehmensnamen auftreten. Der Unternehmensname muß mit der `ConvertHtmlLiterals`-Funktion bearbeitet werden, da viele Firmennamen Sonderzeichen enthalten (wie beispielsweise AT&T). Die Funktion ruft danach `FindNextContent` auf, um die verschiedenen Kurs-Informationen zu bekommen.

Wie Sie in diesem Beispiel sehen können, bietet nicht jeder Dienst alle Informationen an, die das `StockQuote`-Objekt nutzen könnte. Sie können nach anderen Diensten suchen oder mehrere Dienste nacheinander aufrufen, um so viele Informationen wie möglich zu sammeln.

Was würde passieren, wenn Schwab oder Yahoo das Format der Seiten ändern würden? Solange es keine umfangreichen Änderungen sind, bestehen gute Chancen, daß lediglich diese Funktionen zu ändern wären. Was würde geschehen, wenn Schwab oder Yahoo den Kurs-Dienst einstellen würden und wir den Server auf eine andere Web-Site umleiten müßten? Wir müßten eine neue Parser-Funktion schreiben und die Abfragezeile im Formular `frmHolder` ändern. Das wäre alles.

Natürlich könnte der Server leicht modifiziert werden, um mehrere Kurs-Dienste zu verarbeiten und auch alternative Dienste abzufragen, wenn einer vorübergehend ausfallen sollte. Der größte Teil der Arbeit ist hiermit bereits getan – den Rest überlasse ich Ihnen.

### 15.2.7 Referenzierung

Eine Kleinigkeit bei der Funktionsweise dieses Servers betrifft die Referenzierung. Was geschieht, wenn beispielsweise ein Client geschlossen wird, während seine Abfrage noch in Bearbeitung ist?

Erstaunlicherweise bereitet so ein Fall keinerlei Probleme. Zum einen hält das Objekt eine Referenz auf das Rückruf-Objekt, während die Abfrage bearbeitet wird, was zunächst schon einmal den Client vom Terminieren abhalten kann. Zum anderen kann (und sollte) ein Client die Methode `CancelNotification` aufrufen, um anzuzeigen, daß er die Abfrage abbrechen und schließen möchte. Außerdem braucht ein Client keinen Rückruf zu verwenden. Er kann auch `Nothing` an die `GetQuote`-Methode übergeben und fortlaufend die `State`-Eigenschaft überwachen, um festzustellen, wann die Abfrage erledigt ist.

Was passiert nun, wenn der Client terminiert? Dies eliminiert eine Referenz im `StockQuote`-Objekt. Solange jedoch eine Anfrage ansteht, wird die Referenz nicht gelöscht, da sie ja noch von der Collection der `QuoteEngine` referenziert wird. Erst wenn die Abfrage abgeschlossen ist, terminiert das Objekt.

Das gleiche Prinzip sorgt dafür, daß das Formular `frmHolder` entladen wird, wenn keine Abfragen mehr anstehen. Wenn das Formular ständig geladen bleiben würde, könnte der Server niemals terminieren.

### 15.2.8 Das Projekt StockMon.vbp

Das Projekt `StockMon.vbp` ist ein ziemlich simples Projekt, das die ursprüngliche von mir gedachte Aufgabe erfüllt: ein Programm, das eine Auswahl an Kursnotierungen verfolgt und mich periodisch auf Veränderungen aufmerksam macht. Sie finden das leicht verständliche Projekt im Ordner zu Kapitel 15 auf der Buch-CD.

Damit schließen wir nun die langerwartete `StockQuote`-Komponente ab. Wie Sie gesehen haben, beruht sie auf einer ganzen Reihe von ActiveX-Features, die in diesem Teil des Buches beschrieben worden sind.

Wie nahezu jede COM-Code-Komponente operiert auch der `StockQuote`-Server in erster Linie im Hintergrund. Es wird Zeit, daß wir uns mit einem Komponenten-Typ befassen, die vielleicht etwas interessanter ist, und auch eine größere Herausforderung darstellt: den ActiveX-Controls.