Ordner und Dateien auslesen

In vorherigen Kapiteln haben Sie sich hauptsächlich um das Oberflächendesign gekümmert. Sie haben die verschiedensten Steuerelemente kennen gelernt und den Umgang mit ihnen verstanden. Programmiert wurde bisher aber nur ganz wenig. In diesem Teil des Tutorials geht es hauptsächlich um die Beschaffung von verwendbaren Daten.

Im Laufe des Tutorials soll ja eine Art Explorer entstehen. Auf der linken Seite soll die Ordnerstruktur Ihres Rechners erscheinen, links dann die passende Liste von Dateien zum aktuell selektierten Ordner. Wir werden dazu zum ersten Mal zusätzliche Hilfsmittel aus dem .NET-Framework verwenden.

Die erste Überlegung, die Sie machen müssen, betrifft das ereignisgesteuerte Arbeitsmodell von Windows-Anwendungen. Sie müssen sich entscheiden, zu welchem Zeitpunkt Sie beginnen, den Ordnerbaum aufzubauen. Das soll natürlich direkt beim Programmstart passieren, weshalb Sie sich am besten an das **Load**-Ereignis Ihres Formulars halten. Markieren sie dazu das Formular und wählen Sie **Load** im Ereignisfenster mit einem Doppelklick aus. Wie Sie es auch schon aus den anderen Kapiteln kennen, wird für Sie nun die passende Methode erzeugt, in der Sie Ihren Code für dieses Ereignis hinterlegen können. An dieser Stelle sorgen wir dafür, dass der Baum mit Verzeichnisinformationen gefüllt wird.

Laufwerke auflisten

Grundlage für einen guten Explorer ist die Auflistung der Laufwerke als Grundlage für alle weiteren Infos. An diese Information gelangen Sie mit Hilfe einer Klasse des .NET-Frameworks. Damit Sie diese Klasse aber verwenden können, müssen Sie zuvor den benötigten Teil der Bibliothek importieren. Sie tun das, indem Sie mit dem Schlüsselwort `Imports` den Teil nennen, den Sie verwenden wollen. Sie benötigen für Zugriffe auf Ihre Datenträger den Bereich des Frameworks, der für die Zugriffe auf Ein- und Ausgabegeräte zuständig ist. Dieser Teil ist im Bereich `System.IO`. Man nennt diese eingeteilten Bereiche auch *Namespaces*.

Fügen Sie für den Import einfach die folgende Zeile ganz oben in Ihre Quelltext-Datei ein.

```
Imports System.IO
```

Danach steht Ihnen eine Reihe von zusätzlichen Befehlen zur Verfügung, um unter anderen auch die Liste von Verzeichnissen, Dateien und Laufwerken auszulesen – also alles was Sie für die Erstellung des Explorers benötigen.

Fügen Sie nun in der neu erstellen Methode auch folgenden Code ein:

```
Dim drives = DriveInfo.GetDrives()
For Each drive As DriveInfo In drives
    Dim node = TreeView1.Notes.Add(drive.Name)
    node.Tag = drive.RootDirectory
Next
```

Die erste Zeile erstellt eine neue Variable, um das Ergebnis des nachfolgenden Befehls zu speichern. `DriveInfo.GetDrives()` liefert Ihnen eine Liste der Laufwerke Ihres PCs und speichert diese in `drives`. Die anschließende `For Each`-Schleife durchläuft jeden Eintrag der Liste und speichert den aktuellen Listeneintrag in der Variable `drive`, die nur innerhalb der Schleife verfügbar ist. Bei jedem Schleifendurchlauf wird dann ein neuer Eintrag in den Baum erzeugt (standardmäßig werden neue Steuerelemente immer durchnummeriert, daher der Name „TreeView1“). Die Methode Add erwartet von Ihnen die Angabe des Textes, der im Baumeintrag zu lesen sein wird. Mit `drive.Name` wird der Name des Laufwerks hinterlegt. Gleichzeitig gibt diese Methode aber auch den neuen Baumeintrag zurück, den Sie in die Variable `node` speichern, um weiter damit zu arbeiten. In der folgenden Zeile greifen Sie nämlich erneut auf den neuen Eintrag zu, indem Sie die Eigenschaft Tag füllen. In Tag können Sie als Programmierer beliebige Dinge ablegen, um später – vielleicht bei der Selektion von Einträgen – weiterarbeiten zu können. In diesem Beispiel speichern Sie eine Referenz auf das Quellverzeichnis Ihres Laufwerks, um bei der späteren Selektion des Laufwerks schneller auf eine Liste der Dateien zugreifen zu können.

Verzeichnisse rekursiv auflisten

Bisher kann Ihre Anwendung nur Laufwerke auflisten. Um aber auch die enthaltenen Verzeichnisse richtig anzuzeigen, müssen Sie diese auf ähnliche Weise auflisten und in den Baum einfügen. Jeder dieser Ordner kann aber wiederum Verzeichnisse enthalten, und diese wiederum noch mehr Verzeichnisse, usw. Solche verschachtelten Dinge löst man in einfachen Fällen mit rekursiven Funktionsaufrufen. Das bedeutet, dass eine Methode die Sie erstellt haben, sich innerhalb dieser Methode nochmal selbst aufruft, um die selbe Aufgabe mit einem tiefer verschachtelten Verzeichnis nochmal durchzuführen.

Wir werden nun eine eigene Methode erzeugen, die sich genau um die Auflistung einer einzigen Verzeichnisebene kümmert. Dazu brauchen wir 2 Parameter: Der erste übergibt den Baumeintrag, in dem die neuen Verzeichnisse eingefügt werden sollen und der zweite Parameter erhält das Verzeichnis, aus dem die Liste der Unterverzeichnisse entnommen werden soll. Diese Methode fügen Sie am besten direkt unter der **Form1_Load**-Methode ein.

```
Private Sub FillNode(ByVal node As TreeNode, _
    ByVal directory As DirectoryInfo)

    Dim directories = directory.GetDirectories()
    For Each subdirectory As DirectoryInfo In directories
        Dim subnode = node.Nodes.Add(subdirectory.Name)
        subnode.Tag = subdirectory
        FillNode(subnode, subdirectory)
    Next
End Sub
```

Sie erkennen sicher große Ähnlichkeit mit der Methode für die Auflistung der Laufwerke. `directory` in der ersten Zeile repräsentiert ein beliebiges Verzeichnis und liefert alle Unterverzeichnisse mit der Methode `GetDirectories()`. Diese Auflistung speichern Sie in der Variables `directories` und durchlaufen alle Einträge in einer Schleife. Innerhalb der Schleife erzeugen Sie einen neuen Baumeintrag innerhalb des Eintrages, der Ihnen übergeben wurde und

speichern das betroffene Verzeichnis wieder in `Tag`. Neu ist die darauf folgende Zeile, in der Sie die gerade eben erstellte Methode nochmal aufrufen. Dies nennt man Rekursion. Die Methode ruft sich selbst mit anderen Parametern – sprich mit jedem der Unterverzeichnisse – auf, um sich so in die Tiefen des Verzeichnisbaumes zu arbeiten. Die Methode ruft sich somit solange selbst auf, bis sie jeden Ast des Verzeichnisse Ihres Rechners erreicht und in den Verzeichnisbaums eingetragen hat.

Fehlt nur noch der erste Aufruf der Methode, den Sie nun noch in die **Form1_Load**-Methode einfügen müssen. Der beste Platz dafür ist direkt nach der Zeile, in der Sie RootDirectory im Tag des Nodes speichern. Mit der folgenden Zeile starten Sie die Rekursion.

```
FillNode(node, drive.RootDirectory)
```

Der Start der Anwendung dauert jetzt etwas länger, weil das Programm zu Beginn erst alle Verzeichnisse Ihres Rechners durchläuft und in den Baum einfügt, bevor das Fenster angezeigt wird.

Zugriffsprobleme unterbinden

In den meisten Fällen wird diese Anwendung beim Start nun Probleme machen. Ursache dafür sind Verzeichnisse, auf die Sie keinen Zugriff haben. Das System legt auf Ihrer Festplatte Ordner an, für die der Zugriff normalerweise gesperrt ist. Wenn Sie versuchen auf solche Bereiche zuzugreifen, erhalten Sie Fehlermeldungen. Da Ihnen das aber bewusst ist, können Sie mit Hilfe einiger Zeilen Code diese Fehler unterdrücken. Seien Sie sich aber bewusst, dass Sie damit auch alle anderen möglichen Fehler unterdrücken.

Fügen Sie die folgenden Zeilen Code zu Ihrer Methode FillNode hinzu und schließen Sie den bereits vorhandenen Code passend ein, damit Fehlermeldungen bei der Auflistung der Verzeichnisse unterdrückt werden.

```
Try

    [bisheriger Code]

Catch ex As Exception
End Try
```

Dateien auflisten

Ihre Anwendung sollte nun problemlos starten und nach einigen Sekunden Suchzeit für die Verzeichnisse auch den vollständigen Verzeichnisbaum auf der linken Seite abbilden. Leider passiert bei der Selektion eines Verzeichnisses noch nichts, da wir noch nicht definiert haben, was in diesem Fall geschehen soll. Ein guter Datei-Explorer soll bei der Auswahl eines Verzeichnisses links im rechten Bereich eine Liste der Dateien auflisten, die in diesem Verzeichnis enthalten sind.

Wieder stehen wir vor der Frage, an welcher Stelle des Programms nun Code notwendig ist. Immer dann, wenn der User einen neuen Baumeintrag markiert, soll die Dateiliste rechts neu aufgebaut werden. Wir benötigen also das Ereignis, das bei der Änderung der Selektion des Baums eintritt. Verwenden Sie dazu am besten das Ereignis **AfterSelection** des Baums.

Was als Nächstes zu tun ist, können Sie nun vielleicht vermuten. Sie benötigen eine Liste der Dateien, die in einem Verzeichnis vorhanden sind. Diese Liste erhalten Sie natürlich vom Verzeichnis. Und nun kommt der Inhalt von Tag ins Spiel. Sie erinnern sich sicher, dass wir in der Eigenschaft Tag des Nodes im Baum das Verzeichnis gespeichert haben das es repräsentiert.

```
Try
    Dim directory As DirectoryInfo
    directory = TreeView1.SelectedNode.Tag
    Dim files = directory.GetFiles()
    ListView1.Items.Clear()
    For Each file As FileInfo In files
        ListView1.Items.Add(file.name)
    Next
Catch ex As Exception
End Try
```

Wieder umklammern Sie Ihren vollständigen Code mit `Try` und `Catch`, weil auch beim Zugriff auf Dateien Fehler passieren können. Anschließend holen Sie das zuvor gespeicherte Verzeichnis wieder aus der **Tag**-Eigenschaft des Baumeintrags. **SelectedNode** enthält dabei den gerade markierten Eintrag des Baumes. Der darauffolgende Teil ähnelt sehr der Vorgehensweise bei Laufwerken und Verzeichnissen – ist Ihnen also bereits bekannt. Einziger Unterschied: Sie müssen die Dateiliste vor jedem Füllvorgang immer wieder leeren, da von vorherigen Verzeichnisanzeigen noch Dateieinträge vorhanden sind. Das geschieht mit dem *Clear()*-Befehl der **Items**-Auflistung.

Wenn alles richtig läuft, sollte der Explorer ab jetzt die Baumansicht links korrekt aufbauen und auch bei der Selektion von Einträgen die enthaltenen Dateien rechts auflisten. Ein hartes Stück Arbeit, aber im nächsten Kapitel wird es wieder etwas leichter – versprochen.