



Arbeiten mit VC++

# Der individuelle FTP-Client

FTP-Transfers können mit dem normalen Client-Programm von Windows oder mit einem der komfortableren Tools wie WS\_FTP absolviert werden. Man kann auch **eigene FTP-Clients programmieren**.

THOMAS WÖLFER

Die Entwicklung eines vollständigen FTP-Clients mit allem Drum und Dran ist ein etwas aufwändiges Projekt, das den Rahmen dieses Beitrages sprengen würde. Trotzdem werden Sie im Folgenden alles implementieren, was Sie für ein solches Programm benötigen. Das fertige Programm wird in der Lage sein, den Inhalt eines FTP-Servers in einem Explorer-ähnlichen Fenster anzuzeigen, und es wird den Transfer von lokalen Dateien zum FTP-Server per Drag&Drop unterstützen.

Der FTP-Client ist dabei als eine auf eine Dialogbox basierende MFC-Anwendung ausgelegt. Auf der Dialogbox befinden sich einige Edit-Controls, die den Zweck erfüllen, dass der Namen des FTP-Servers sowie ein Benutzername und ein Passwort angegeben werden können. Zusätzlich zu diesen Elementen gibt es noch ein Tree-Control, das den Inhalt des FTP-Servers anzeigen soll.

Diese Dialogbox unterstützt außerdem Drag&Drop für Dateien. Das bedeutet, dass Sie Dateien aus dem Windows-Explorer direkt auf der Dialogbox fallen lassen können. Diese Dateien werden dann auf das gerade ausgewählte Verzeichnis auf dem FTP-Server transferiert. Dabei ist diese Funktionalität aus Gründen des benötigten Umfangs nicht vollständig bis ins letzte Detail optimal implementiert. Sie erhalten

aber alles an Grundwissen, um das Programm passend weiter auszubauen. Bei der Gelegenheit sei noch erwähnt, dass das auf der CD befindliche Programm Dateien nicht richtig kopiert. Statt dessen wird nur per Dialogbox angezeigt, welche Aktion ausgeführt werden würde. Wenn Sie tatsächlich Dateitransfers durchführen möchten, müssen Sie den Quell-Code übersetzen und dabei eine Präprozessor-Direktive setzen.

## ■ Drag&Drop mit Dateien

Kommen wir nun zur Implementierung. Zunächst legen Sie ein ganz normales MFC-Projekt neu an, und verwenden als Vorlage eine Anwendung, die auf eine Dialogbox basiert.

Die erste Problematik ergibt sich durch die Tatsache, dass die Dateien per Drag&Drop kopiert werden sollen. Wenn Sie Dateien mit dieser Methode auf einem Fenster fallen lassen, erhält die Anwendung bzw. das betroffene Fenster die WM\_DROPFILES-Nachricht. Für diese sieht der Class-Wizard bei ei-

ner Dialogbox normalerweise keinen Handler vor. Dies lässt sich einfach auflösen, denn die Programmierer des Class-Wizards haben diese Problemstellung vorhergesehen. Sie können den verwendeten Message-Filter der Class-Wizards verändern. Wenn Sie statt des normalen Dialogbox-Filters den einer anderen Fensterklasse verwenden, erscheinen die zusätzlich möglichen Handler.

Das alleine reicht noch nicht, denn ein Fenster erhält von Haus aus keine WM\_DROPFILES-Nachrichten – völlig gleichgültig, ob ein Handler dafür definiert wurde oder nicht. Damit diese Nachricht an ein Fenster versendet wird, muss das Fenster selbst erst einmal signalisieren, dass es bereit ist, WM\_DROPFILES-Messages zu empfangen. (Warum das für diese Nachricht so ist, während fast alle anderen Nachrichten anders behandelt werden, wird vermutlich für immer ein ungelöstes Design-Rätsel von Windows bleiben.)

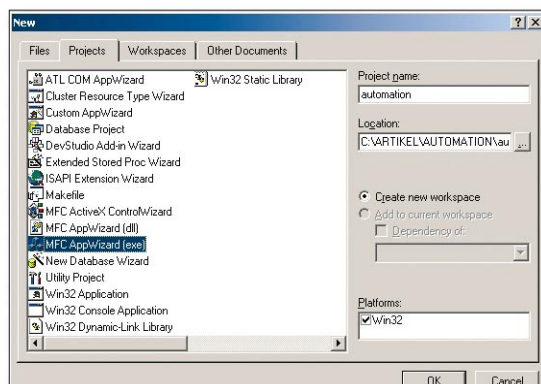
Dieses Signal können Sie im Quell-Code auslösen. Sie müssen die Methode *DragAcceptFiles()* der CWnd-Klasse verwenden oder die *DragAcceptFiles()*-Funktion aus dem SDK aufrufen.

Bei Dialogboxen ist die Sache deutlich einfacher möglich. Solche Fenster haben ein Style-Bit mit dem Namen „Accept Files“. Wird dieses Bit gesetzt, empfängt das Fenster auch die WM\_DROPFILES-Nachrichten, ohne dass *DragAcceptFiles()* explizit von Ihnen aufgerufen werden müsste. Das Bit können Sie im „Properties“-Dialog der Dialogbox setzen, indem Sie die passende Option einschalten.

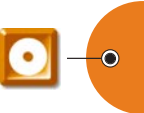
Kommen wir nun zur Implementierung des Message-Handlers.

```
void CFTPDlg::OnDropFiles(HDROP  
    ↳ hDropInfo)  
{  
    SetActiveWindow();  
    UINT nFiles = ::Drag  
    ↳ QueryFile(hDropInfo,  
    ↳ (UINT)-1, NULL, 0);  
  
    for (UINT iFile = 0; iFile  
    ↳ < nFiles; iFile++)  
    {  
        TCHAR szFile  
        ↳ Name[ _MAX_PATH];  
        ::DragQueryFile(hDrop  
        ↳ Info, iFile, szFile  
        ↳ Name, _MAX_PATH);  
        OnDroppedFile  
        ↳ (szFileName);  
    }  
    ::DragFinish(hDropInfo);  
  
    CDialog::OnDropFiles  
    ↳ (hDropInfo);  
}
```

Der Drag&Drop-Support für Dateien wird mit einem speziellen Handle vom Typ HDROP unterstützt. Dieses



FÜR DAS FTP-CLIENT-PROJEKT benötigen Sie ein Projekt für eine MFC-Dialog-Box-Anwendung.



Handle müssen Sie an die Spezialfunktionen für die Nutzung der dahinter liegenden Informationen weiterreichen. Tritt ein WM\_DROPFILES-Ereignis ein, ermitteln Sie zunächst mit der *DragQueryFile()*-API die Anzahl der fallengelassenen Dateien.

Danach können Sie eine Schleife über die soeben ermittelte Anzahl an Dateien programmieren. In jedem Schleifendurchlauf fragen Sie dabei mit *DragQueryFile()* Informationen über eine spezielle Datei ab. Als Antwort auf Ihre Informationsanfrage erhalten Sie den kompletten Pfad zur Datei zurück.

Ist die Schleife abgearbeitet, müssen Sie Windows mit der *DragFinish()*-API noch signalisieren, dass Sie das HDROP-Handle nicht länger benötigen.

Im Zuge der Schleife rufen Sie noch eine selbst programmierte Funktion namens *OnDroppedFile()* auf, der Sie als Parameter den Pfad zur Datei übergeben. Diese Funktion kümmert sich um die Bearbeitung einer einzelnen Datei. *OnDroppedFile()* hat folgenden Aufbau:

```
void CFtpDlg::OnDroppedFile(const
    ↳char * psz)
{
    if( m_pFtpConnection !=
        ↳NULL)
    {
        char drive[_MAX_DRIVE];
        char dir[_MAX_DIR];
        char fname[_MAX_FNAME];
        char ext[_MAX_EXT];
        _splitpath( psz, drive,
            ↳dir, fname, ext );
        CString strTarget =
            ↳m_FtpTreeCtrl.Get
            ↳CurrentDirectory() +
            ↳"/" + CString( fname)
            ↳+ CString( ext);

        // files werden nicht wirklich
        ↳kopiert solange
        // dieses define nicht gesetzt
        ↳ist...
        #ifdef _DO_TRANSFER_FILES
            m_pFtpConnection->Put
                ↳File( psz, strTarget);
        #else
            AfxMessageBox( "
                ↳(Simulation!):
                ↳Übertrage Datei von:
                ↳" + CString(psz) +
                ↳" nach: " + str
                ↳Target);
        #endif
    }
}
```

Grundsätzlich arbeitet *OnDroppedFile()* nur dann, wenn der Zeiger auf die FTP-Verbindung, die zur Dialogbox-Klasse gehört, auch gesetzt ist. Woher dieser Zeiger stammt, erfahren Sie später in diesem Beitrag.

Ist der Zeiger gesetzt, müssen Sie den Pfad der Datei auseinandernehmen, um den Namen und die Erweiterung der Datei zu erfahren. Diese Information ist

notwendig, weil Sie für das Zusammenbauen des Zielpfades auf dem FTP-Server wichtig ist. Der andere Teil dieses Pfades stammt aus einem in der Dialogbox eingebetteten Control, dem *FtpTreeCtrl*. Auch über dieses Control erfahren Sie später mehr.

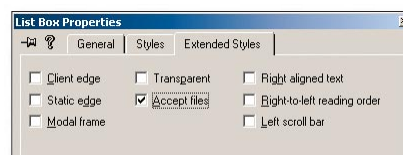
Ist der Pfad zusammengebaut, können Sie den Zeiger auf die FTP-Verbindung einfach verwenden, um die Datei zu übertragen. Wie im abgedruckten Quell-Code zu sehen, enthält das Programm an dieser Stelle eine Präprozessor-Anweisung, die verhindert, dass der Transport der Datei wirklich durchgeführt wird. Das ist im Rahmen des Programmierens sinnvoll, denn dieses Vorgehen beschleunigt die Arbeit mit dem Programm. In der „fertigen“ Version des Programms brauchen Sie nur das Symbol *\_DO\_TRANSFER\_FILES* zu definieren, um den Kopiervorgang durchzuführen.

Soviel zum Drag&Drop-Support, kommen wir nun zum eigentlich relevanten Internet-Teil des Programms. Für die Arbeit mit dem Internet bieten die MFC eine ganze Reihe von Klassen. Die zentrale Klasse dabei ist die „CInternetSession“. Diese Klasse kümmert sich um den Aufbau einer Verbindung mit dem Internet und ist auch in der Lage FTP, HTTP, oder Gopher-Sessions zu erzeugen.

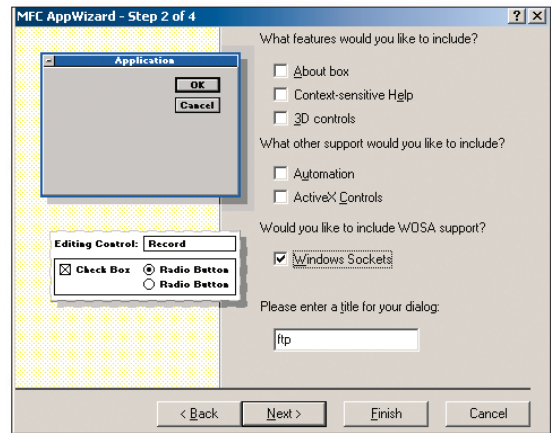
Im Zuge des FTP-Clients brauchen Sie eine FTP-Session. Um diese aufzubauen sind drei Informationen notwendig:

- Der Name des FTP-Servers, der kontaktiert werden soll,
- der User-Name, den Sie zum Anmelden am Server benötigen,
- das zum User-Name passende Passwort.

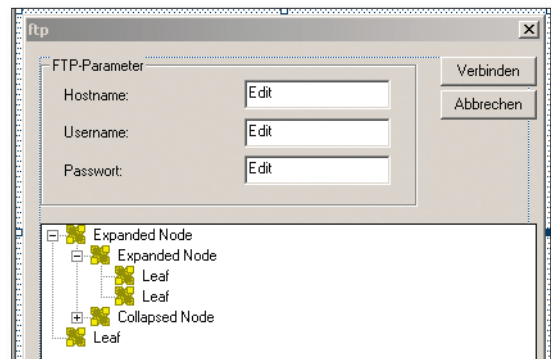
Diese Informationen lesen Sie zunächst



**DIE OPTION ACCEPT FILES** müssen Sie einschalten, damit Sie die WM\_DROPFILES-Nachricht empfangen können.



**DER SOCKETS-SUPPORT** ist wichtig, da die Internet-Kommunikation auf den Sockets aufsetzt.



**DAS BILD ZEIGT DIE DIALOGBOX** des FTP-Clients im Design-Mode des Visual-Studios.

aus der Dialogbox aus und speichern sie in passenden Variablen:

```
CString strHostname;
GetDlgItemText( IDC_HOSTNAME,
    ↳strHostname);
CString strUsername;
GetDlgItemText( IDC_USERNAME,
    ↳strUsername);
CString strPassword;
GetDlgItemText( IDC_PASSWORD,
    ↳strPassword);
```

Danach können Sie eine Internet-Session erzeugen. Im Betrieb des Programms müssen Sie dabei berücksichtigen, dass sich die CInternetSession-Klasse nicht um die tatsächliche Verbindungsaufnahme mit dem Internet kümmert. Die Leitung zu Ihrem ISP muss bereits geöffnet sein. Das ist weiter nicht dramatisch, da alle aktuellen Windows-Versionen über Mechanismen verfügen, die dies automatisieren. Im besten Fall sollte daher der Versuch eine CInternetSession zu starten, automatisch dazu führen, dass der Windows Dialer gestartet wird und Ihren ISP anruft.

Die Internet-Session wird beim Beispielpogramm in der *OnInitDialog()*-Methode der Dialogbox angelegt. So lange die Dialogbox geöffnet bleibt, steht auch die Internet-Session zur Verfü-



gung. Damit Sie diese auch verwenden können, wird der Zeiger auf diese Session in einer Member-Variablen der Dialogbox-Klasse gespeichert:

```
m_pInetSession = new CInternet
    ↳Session();
```

Mit Hilfe dieses Zeigers können Sie die FTP-Verbindung erzeugen. Für diese Verbindung brauchen Sie die zuvor ermittelten Angaben über den FTP-Server:

```
try
{
    m_pFtpConnection = m_pInet
    ↳Session->GetFtpConnection
    ↳(strHostname, strUsername,
    ↳strPassword);
}
catch (CInternetException* pEx)
{
    // wininet library fehler ?
    TCHAR szErr[1024];
    if (pEx-
    ↳->GetErrorMessage(szErr,
    ↳1024))
    ↳AfxMessageBox(szErr,
    ↳MB_OK);
    else AfxMessageBox
    ↳("Unerwarteter Fehler",
```

```
↳MB_OK);
pEx->Delete();
m_pFtpConnection = NULL;
}
```

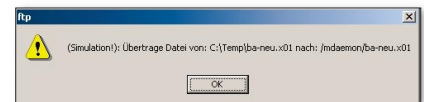
Nachdem der Aufbau einer FTP-Verbindung einer Vielzahl von Dingen scheitern kann, ist es sinnvoll, den Versuch des Verbindungsaufbaus mit einem *try/catch*-Block zu umgeben. Tritt eine Exception auf, wird die Kontrolle an den *Catch*-Block übergeben, sodass Sie den Grund für den Fehlschlag herausfinden können.

Zu diesem Zweck bieten die MFC die *CInternetException*-Klasse. Diese Klasse können Sie mit der Methode *GetErrorMessage()* nach einer als String vorliegenden Begründung für den Fehlschlag fragen. Meist ist dies einfach ein falsches Passwort oder Ähnliches. Es kann aber auch sein, dass keine Verbindung zum FTP-Server hergestellt werden konnte.

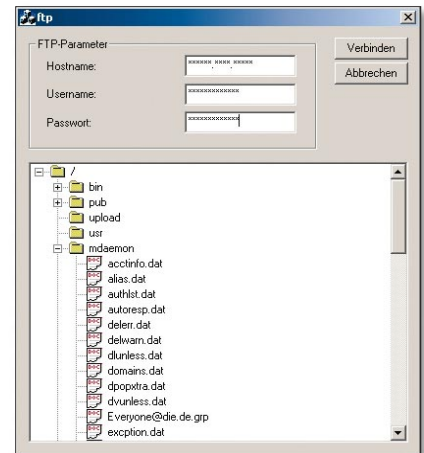
*GetFtpConnection()* erhält für den Aufbau der Verbindung drei Parameter, mit denen der Zielrechner und die Authentifizierungs-Daten spezifiziert werden.

Bei der Benutzung von *GetFtpConnection()* sollte Ihnen bewusst sein, dass das FTP Protokoll nie dafür gedacht war, von Software automatisiert verwendet zu werden. Bei FTP wird davon ausgegangen, dass ein Mensch die Meldungen des FTP-Servers liest und dann passende Antworten gibt. Das führt dazu, dass *GetFtpConnection()* zwar mit vielen, aber nicht mit allen FTP-Servern Verbindungen aufnehmen kann. Gibt ein FTP-Server „ungewöhnliche“ Meldungen von sich, scheitert die Verbindungsaufnahme. Das *GetFtpConnection()* hat keine Chance, die passende Stelle für das Übertragen der Benutzer- und Passwort-Daten zu finden. Mit den FTP-Servern von Microsoft gibt es keine Probleme und auch die fast überall eingesetzten BSD-Server funktionieren tadellos. Ungewöhnlichere Server hingegen sind nicht immer ohne weiteres anzusprechen.

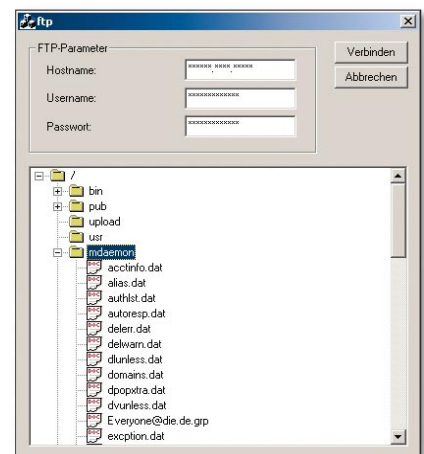
Wichtig ist noch ein Hinweis auf eine andere Einschränkung der FTP-Klasse von MFC. Handelt es sich beim entfernten FTP-Server um einen Unix-Server und werden auf diesem Server



**DAMIT DIE SACHE** schneller geht, wird der Transfer der Dateien nur simuliert. Das Programm zeigt eine entsprechende Meldung an.



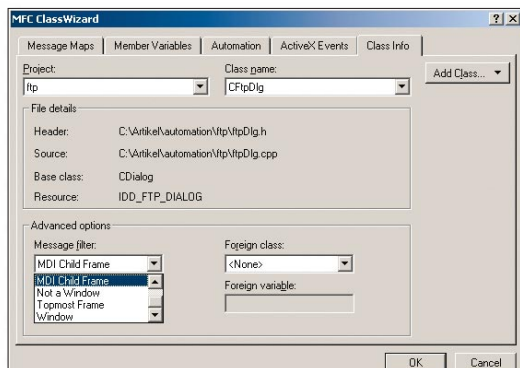
**DER FTP-CLIENT IN ACTION:** Der Inhalt des Servers wird aufgelistet. Sie können sich außerdem die einzelnen Verzeichnisinhalte anzeigen lassen.



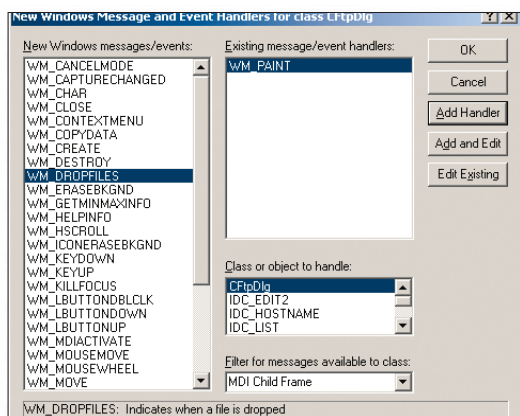
**DAS AKTUELLE VERZEICHNIS** auf dem FTP-Server wird gesondert markiert. In dieses Verzeichnis werden Dateien kopiert, die per Drag&Drop auf dem Client fallen gelassen werden.

Links verwendet, um Verzeichnisse innerhalb des FTP-Verzeichnisbaums einzublenden, schlagen einige Funktionen fehl. Diese eingblendeten Verzeichnisse werden zwar erkannt, allerdings gelingt es nicht, Dateien, die sich innerhalb solcher Verzeichnisse befinden, aufzulisten – zumindest nicht ohne weiteren Aufwand.

Könnte eine Verbindung hergestellt werden, übergeben Sie diese an das bereits erwähnte *FtpTreeCtrl*, das die eigentliche Arbeit fürs FTP erledigt.



**IM CLASSWIAZRD** können Sie verschiedene Nachrichten-Filter einstellen. Damit Sie per ClassWizard einen Handler für *WM\_DROPFILES* definieren können, müssen Sie den normalerweise für Dialoge verwendeten Filter ändern.



**NACHDEM SIE DEN MESSAGE-FILTER** im ClassWizard verändert haben, bietet dieser an, die *WM\_DROPFILES*-Nachricht zu behandeln.



## Das FTP-Tree-Control

Das FTP-Tree-Control erzeugen Sie mit dem ClassWizard. Dabei legen Sie eine neue Klasse auf Basis der MFC *CTreeCtrl*-Klasse an. Das *CTreeCtrl* behandelt abstrahiert ein Windows TreeControl. Nachdem der Inhalt der FTP-Site in einem Baum dargestellt werden soll, ist dies die perfekte Basisklasse für den FTP-Client.

Die von *CTreeCtrl* abgeleitete Klasse erweitern Sie um zwei Funktionen: *PopulateTree()* und *ListDirectory()*. Die erste dieser beiden Funktionen füllt den Baum auf, die zweite dient dazu, nachträglich die Daten für ein bestimmtes Verzeichnis in den Baum einzutragen.

*PopulateTree()* iteriert dabei im Wesentlichen über den Baum und ruft die *Expand()*-Methode des *CTreeCtrl* auf, wenn ein Verzeichnis gefunden wird – zum ersten Mal beim ersten Auffüllen des Baumes anhand des Root-Verzeichnisses auf dem FTP-Server. Jedesmal wenn *Expand()* aufgerufen wird, löst das die ITEMEXPANDING-Nachricht aus, die Sie mit dem passenden Handler, den Sie mit dem Class-Wizard einfügen können, behandeln. Dieser Handler selbst tut nicht viel. Er ermittelt nur das aktuelle Verzeichnis im Baum und ruft die *ListDirectory()*-Funktion auf. Diese ist die Instanz, die sich tatsächlich um die Kommunikation mit dem FTP-Server kümmert, denn diese Funktion muss den Inhalt des vorgegebenen Verzeichnisses ermitteln und im Baum eintragen.

Dazu verwendet die Funktion die *CFileFind*-Klasse der MFC. Wenn Sie schon einmal ein rekursives Dateisuchprogramm mit MFC programmiert haben, kennen Sie vermutlich die *CFileFind*-Klasse.

Diese kann zum rekursiven Durchsuchen des Dateisystems auf der Festplatte verwendet werden. In jeder Rekursion wird dabei ein tiefer gelegenes Verzeichnis abgearbeitet, indem eine neue Instanz der *CFileFind*-Klasse verwendet wird, ganz einfach auf dem Stack.

Die *CFileFind*-Klasse funktioniert sehr ähnlich wie die *CFileFind*-Klasse. Es gibt einen kleinen, aber sehr wichtigen Unterschied: Pro *CFileFind*-Instanz können Sie nur jeweils eine *CFileFind*-Klasse benutzen. Somit ist es unmöglich, den gleichen rekursiven Ansatz zum Iterieren über einen FTP-Server zu verwenden, der mit *CFileFind* auf der lokalen Festplatte so angenehm ist.

Aus diesem Grund muss eine iterative Methode gewählt werden. Beim

## DIE MFC-INTERNET-KLASSEN

Die MFC-Internet-Klassen bedienen die wichtigsten Internet-Protokolle. Dies sind HTTP, FTP und Gopher. Dabei stehen für die einzelnen Verbindungstypen spezielle Verbindungsklassen zur Verfügung. Diese Klassen sind alle von *CInternetConnection* abgeleitet und haben entsprechende Namen: *CFTPConnection*, *CHTTPConnection* und *CGopherConnection*.

Alle Verbindungen können nur mit Hilfe einer Internet-Sitzung hergestellt werden, und diese ist durch die *CInternetSession* abstrahiert. Um eine der Verbindungen herzustellen, müssen Sie zunächst eine Internet-Session erzeugen. Diese bietet die Möglichkeit, Verbindungen eines bestimmten Typs aufzubauen. Darüber hinaus stellt die MFC noch „FileFind“-Klassen für FTP und Gopher bereit, aber nicht für HTTP. Das liegt am Protokoll, denn HTTP bietet keine

Möglichkeit über Dateien auf dem HTTP-Server zu iterieren. Dabei ist es wichtig zu wissen, dass Sie immer nur eine „FileFind“-Instanz pro „Connection“ verwenden können. Die *InternetFind*-Klassen unterscheiden sich an dieser Stelle von der normalen *CFileFind*-Klasse, bei der beliebig viele Instanzen parallel mit dem gleichen Dateisystem verwendet werden können.

Als unterstützende Klasse gibt es noch die *CInternetException*, die als Exception-Objekt für die Behandlung von Ausnahmefehlern im Internet-Bereich verwendet wird. Darüber hinaus bieten die MFC noch einige Hilfsfunktionen an. Dazu gehört zum Beispiel die Funktion *AfxParseURL()*, mit der eine URL untersucht werden kann. *AfxParseURL()* liefert den Typ des Dienstes aus der URL und alle Einzelteile der URL als solches.

*CFileFind* wird das so gelöst, dass zunächst nur über den Inhalt eines Verzeichnisses iteriert wird. Dieser Inhalt wird im Tree-Control eingetragen. Wird beim Eintragen des Inhaltes ein Verzeichnis gefunden, so wird dieser Umstand gemerkt.

Ist der Inhalt des Verzeichnisses abgearbeitet, überprüft *ListDirectory()*, ob in diesem Zuge auch Verzeichnisse – zumindest eines – gefunden wurden. Ist das der Fall, werden diese Verzeichnisse im Nachhinein abgearbeitet.

Die elementare Schleife, die in beiden Fällen verwendet wird, hat den folgenden Aufbau (der Quell-Code, der nur für die Verwaltung des Tree-Controls zuständig ist, wurde hier weggelassen. Im Quell-Code auf der CD finden Sie aber die vollständige Funktion):

```
CFileFind ftpFind(m_pFtp
```

```
Connection);
BOOL fContinue = ftpFind.FindFile
( strSearchDir);
while( fContinue)
{
    fContinue = ftpFind.Find
    NextFile();
    CString strFileName = ftp
    Find.GetFileName();
    // nun ist der Name der
    Datei bekannt.
    If( ! ftpFind.IsDots())
    {
        if( ftpFind.
        IsDirectory())
        {
        }
        else
        {
        }
    }
}
ftpFind.Close();
```

Zunächst wird eine *CFileFind*-Instanz erzeugt. Diese erhält im Kon-

**FTP-SERVER MELDEN SICH** so, wie es dem Programmierer Spaß macht, denn FTP ist eigentlich nicht dafür gedacht, automatisiert zu werden. (Vergleiche oberes und unteres Bild.)



struktur einen Zeiger auf die zu verwendende FTP-Verbindung. Wie bereits erwähnt, können Sie immer nur eine Instanz von *CFtpFileFind* pro *CFtpSession* verwenden. Es spricht aber

Informationen über eine Datei erst abrufbar sind, wenn *FindNextFile()* bereits aufgerufen wurde. Das bedeutet, die *CFtpFileFind*-Instanz enthält erst nach dem ersten Aufruf von *FindNextFile()*

die Daten über die erste Datei – und nicht bereits, wie man vielleicht vermuten könnte, schon nach dem initialen Aufruf von *FindFile()*.

Liegen Daten über eine Datei an, so können Sie die *CFtpFileFind*-Instanz danach befragen. Dazu ist es wichtig zu wissen, dass *CFtpFileFind* beim Iterieren von Verzeichnissen keinen Unterschied zwischen Dateien und Ordnern macht. Beides wird in der Iteration zurückgeliefert. Wenn Sie also wissen müssen – und die *CFtpTreeCtrl*-Klasse muss das wissen – ob es sich bei einer gegebenen Datei um eine Datei oder einen Ordner

handelt, müssen Sie die Methode *IsDirectory()* bemühen. Diese Methode liefert im Falle eines Ordners „true“, im anderen Falle „false“.

Außerdem ist noch ein Sonderfall zu beachten. Jedes Verzeichnis enthält zwei „spezielle“ Verzeichnisse. Das sind die Verzeichnisse, die beim „dir“-Kommando mit einem oder zwei Punkten markiert werden. Dabei handelt es sich um das „Wurzelverzeichnis“ und das direkt oberhalb des aktuellen Verzeichnisses angeordnete Verzeichnis. Bei der Anzeige einer Verzeichnisstruktur in einem Baum sind diese beiden Verzeichnisse nie von Interesse. Schließlich kann man an der Baumstruktur ablesen, wie die „Eltern-Verhältnisse“ der einzelnen Verzeichnisse aussehen. Daher werden diese Verzeichnisse mit der Spezialbehandlung *IsDots()* von vornherein ausgeschlossen.

## ■ Verzeichnisse öffnen

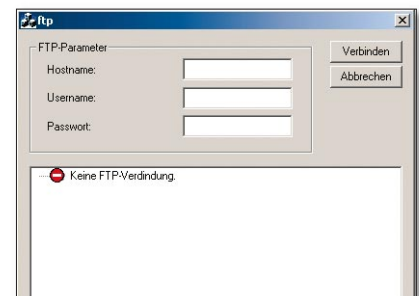
Jedesmal, wenn zur Laufzeit des Programms auf eines der Ordnersymbole geklickt wird, löst dies die *ITEMEXPANDING*-Nachricht aus. Jedesmal wird *ListDirectory()* ausgeführt. Das Programm liest nicht ganz zu anfang den kompletten FTP-Server aus, sondern tut dies nur verzeichnisweise für jeweils das Verzeichnis, das der Anwender soeben geöffnet hat. Für den Drag&Drop-Support im Beispielprogramm wurde ein

weiterer Message-Handler implementiert. *CFtpTreeCtrl::OnSelchanged()* wird immer aufgerufen, wenn die Selection innerhalb des Tree-Controls verändert wurde. Das momentan selektierte Element wird innerhalb des Baumes in einer gesonderten Farbe markiert.

Wird nun der Handler aufgerufen, ermittelt er anhand des aktuell selektierten Elementes den kompletten Pfad zum momentan selektierten Verzeichnis auf dem FTP-Server, und speichert diesen in der Variablen *m\_strCurrent* des *CFtpTreeCtrl*. Das ist genau die Variable, deren Inhalt zurückgeliefert wird, wenn das Control mit *GetCurrentDirectory()* nach dem aktuellen Verzeichnis gefragt wird. Wie dies geschieht, haben Sie



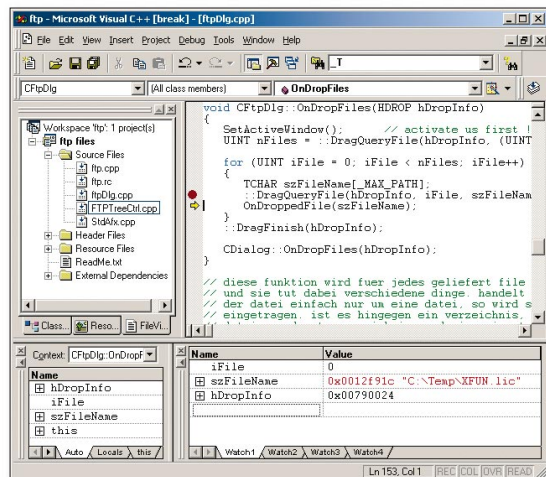
**MANCHMAL LIEFERT DIE CINTERNETEXCEPTION** auch sinnvolle Fehlermeldungen, zum Beispiel dann, wenn der angeforderte Server nicht existiert.



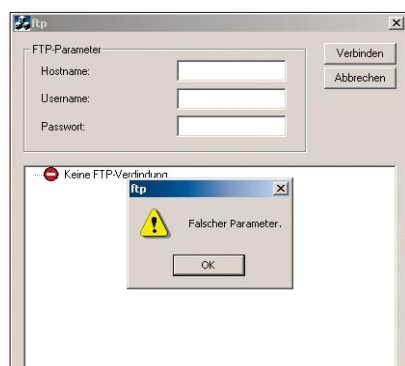
**WENN NOCH KEINE VERBINDUNG** hergestellt wurde, zeigt der FTP-Client eine entsprechende Meldung an.

bereits weiter vorne im Artikel, im Bereich über den Drag&Drop-Support, erfahren.

Mit den Internet- und Dateiklassen der MFC können Sie sehr leistungsfähige auf Internet basierende Programme mit erstaunlich wenig Aufwand programmieren. So bieten diese Klassen, abgesehen vom FTP-Protokoll, unter anderem auch die Möglichkeit, Webseiten direkt herunterzuladen, oder Dateien unter Umgehung des FTP-Protokolls, direkt auf einem Server anzulegen oder auszulesen. Mit dem vorliegenden Beispiel sollte Ihnen das komplette Grundwissen für die Programmierung auch aufwändigerer Internet-Programme zur Verfügung stehen. UR



**IM VISUAL STUDIO DEBUGGER** ist es relativ einfach herauszufinden, welche Informationen bei den *WM\_DROPFILES*-Nachrichten übermittelt werden.



**OHNE DIE BENÖTIGTEN PARAMETER** kann keine FTP-Verbindung hergestellt werden. Es wird eine *CInternetException* ausgelöst und gefangen. Der Grund wird angezeigt, leider ist dieser nicht immer sehr aussagekräftig.

nichts dagegen, dass Sie mehrere *CFtpFileFind*-Instanzen mit mehreren *CFtpSessions* parallel benutzen.

Nun wird die Suche mit dem ersten Aufruf von *FindFile()* gestartet. *FindFile()* erhält dabei einen String-Parameter. Dieser gibt das Verzeichnis und eine Wildcard für die darin zu suchenden Dateien an, also zum Beispiel „/\*“ für alle Dateien im Wurzelverzeichnis.

Könnte eine Datei gefunden werden, so liefert *FindFile()* „true“ und die Iteration kann mit *FindNextFile()* weitergeführt werden. Die Iteration endet, wenn *FindNextFile()* „false“ liefert. Dabei ist es wichtig zu wissen, dass die