Lab 11.3: Using the FTP WinInet Classes

Objectives

After completing this lab, you will be able to use the MFC WinInet classes to:

- Create an Internet session.
- Establish an FTP connection.
- Use FTP to transfer files.
- Close an FTP session.

Prerequisites

You should be able to create MFC applications that invoke and use modal dialog boxes.

You should have completed Chapter 11 through the section titled "Writing FTP Applications."

Lab Setup

This demonstration shows what you will accomplish during the lab.

Estimated time to complete this lab: **60 minutes**.

Exercises

The following exercise provides practice working with the concepts and techniques covered in this chapter.

Exercise 1: Creating an FTP Application

In this exercise, you will:

- Modify the baseline application to invoke a dialog box to get user information (search parameters) required for an FTP request.
- Implement an FTP transfer function that encapsulates the creation, file transfer, and termination of an FTP connection.
- Implement a function to do a recursive directory search and file retrieval using an FTP connection.

Before you start this lab, you should have installed Internet Explorer. You also must have an account with an Internet service provider or access to the Internet via a corporate firewall.

Alternatively, you can test these labs on a standalone computer. To do so, ensure that the Microsoft Personal Web Server (PWS)software is installed on your computer and that the Web server is properly configured and started. For more information on installing and configuring PWS, see the section Microsoft Personal Web Server in Chapter 11 of this course.

Copy the contents of \Labs\C11\Lab03\Baseline to your working directory.

The completed code for these exercises is in \Labs\C11\Lab03\Xxx, where Xxx is the exercise number.

Exercise 1: Creating an FTP Application

It is often useful to copy many files of a given type from an FTP server. The purpose of this lab is to create an application that uses the MFC WinInet classes to create an FTP utility that accepts wildcard strings to retrieve a subset of files existing on an FTP file server.

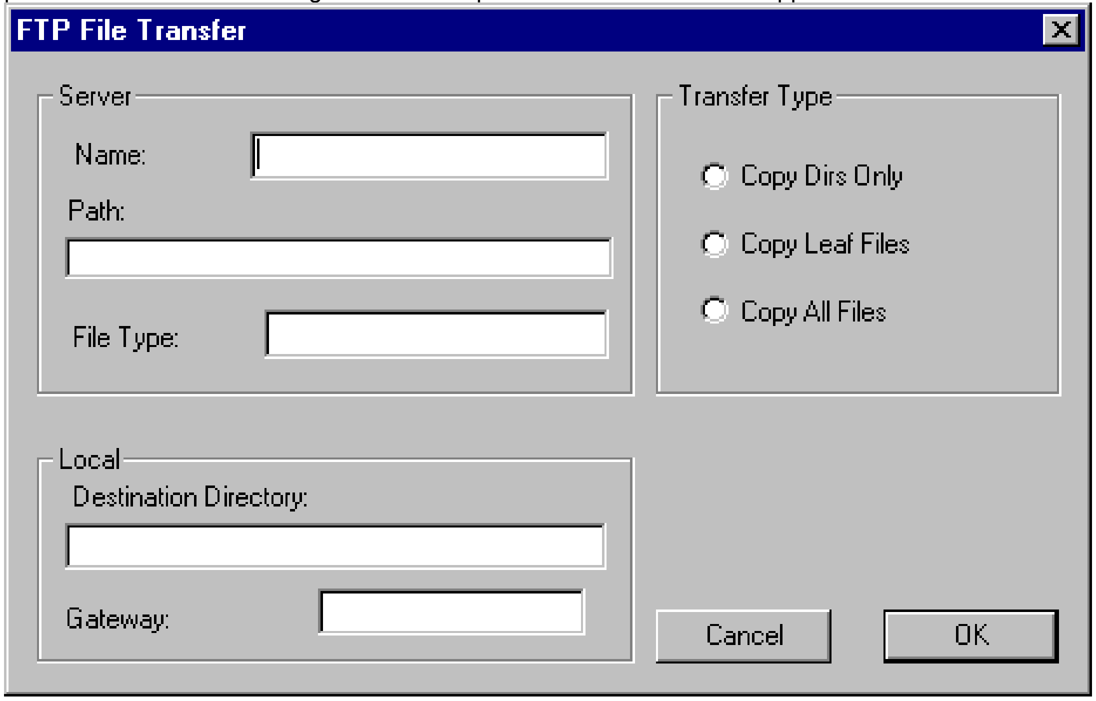
In this exercise, you will:

- Modify the baseline application to invoke a dialog box to get user information (search parameters) required for an FTP request.

- Implement an FTP transfer function that encapsulates the creation, file transfer, and termination of an FTP connection.
- Implement a function to do a recursive directory search and file retrieval using an FTP connection.

About the Baseline Application

The baseline application contains a File menu item "FTP Transfer" with an empty command handler and a dialog box resource and class (**CFtpDlg**) that you can use to query the user for FTP search parameters. This is the dialog box resource provided with the baseline application.



This table lists the member variables in the **CFtpDlg** class that are used to store the FTP search parameters that the user enters.

Member Variable	Data Type	Description
m_strServer	**CString**	This string stores the path for the FTP server.
m_strPath	**CString**	This string stores the path string entered by the user. The string can contain wildcard characters.
m_strFile	**CString**	This string stores the specification for the file, or types of files, to be copied. The string can contain wildcard characters.
m_strRoot	**CString**	This string stores the path to the local destination directory.
m_strGateway	**CString**	This string stores the specification for the gateway, if one is required.
m_nCopyType	**int**	This integer stores a value corresponding to one of three enumerated values that indicates the type of copy to perform: Copy directories only Copy leaf files Copy all (directories and leaf files).

This table lists the **CFtpGet** class members used or implemented in this exercise.

Member Variable or Function	Data Type or Return Type	Description
m_arrStaticPath	**CStringArray**	This string array stores the original path specification, with each element of the array containing one path element. The wildcard characters are kept intact in this array.
m_arrDynamicPath	**CStringArray**	This string array stores the original path specification, with each element of the array containing one path element. The wildcard characters are replaced with actual path elements during the FTP tree traversal.
DoFtpTransfer()	**void**	This public function is called by the menu handler to initialize the FTP connection, to call CopyDirectory() to perform the actual FTP transfer, and to close the FTP connection. Implemented by the student.
CopyDirectory(int PathIndex)	**BOOL**	This member function performs copying of all matching files from one directory of the FTP server. Recursive calls are made to this function to copy files from child directories of the current FTP directory. Implemented by the student.
SetFtpDirectory(int index)	**BOOL**	This member function performs copying of all matching files from one directory of the FTP server. Recursive calls are made to this function to copy files from child directories of the current FTP directory. Implemented by the student.
PathToArray(CString& path, CStringArray& strArray)	**void**	This utility member function takes a path and extracts the path elements and stores them into the string array. Used by the student.
GetLocalDirectory(int level, CString& FullDir)	**BOOL**	This utility member function returns the current local directory, as determined by the directory level parameter. Used by the student.

In addition, the **CFtpView** class contains the **AddToView** helper function, which adds a string to the view's string array and updates the view screen. The **AddToView** function receives a single LPCTSTR argument and has a **void** return type.

Get User Information and Start the FTP Operation

There are four main steps required to invoke a dialog box and start the FTP operation:

1. Include the header files for the **CFtpDlg** dialog box class and the **CFtpGet** class in the view class's implementation file.
2. Create a **CFtpDlg** object and invoke it to query the user for the FTP search parameters.
3. Create a **CFtpGet** object, passing the appropriate dialog box member variables as arguments to the **CFtpGet** constructor.
4. Invoke the **CFtpGet** member function **DoFtpTransferB** to start the transfer.

➢ **Add the required include files**

1. Use the Developer Studio editor to open the file FtpView.Cpp.
2. Add the following directives after the line #include "Stdafx.H":

```
#include "ftpdlg.h"
#include "ftpget.h"
```

➢ **Create and invoke a CFtpDlg object**

Inside the **CFtpView::OnFileFtpTransfer** command handler, after the call to the **OnViewClear** function, add code to create an **CFtpDlg** object. Invoke **DoModal** only if the user clicks OK.

```
// Display dialog box to prompt user for FTP transfer parameters
CFtpDlg dlg(this);
// Do transfer only if user clicked the OK button
if (dlg.DoModal()==IDOK)
```

➢ **Create a CFtpGet object using information from dialog box**

Inside the **DoModal if** block, add code to construct a **CFtpGet** object using the dialog box's search parameters.

```
{                                          // Open DoModal if block
  // Create an instance of the FTP object using dialog box variables
  CFtpGet ftpget(this, dlg.m_strServer, dlg.m_strFile,
          dlg.m_strPath, dlg.m_strRoot, dlg.m_strGateway,
          dlg.m_nCopyType);
```

➢ **Start the transfer by invoking the DoFtpTransfer function**

Add code to do the FTP transfer and close **DoModal if** block.

```
  // Do the transfer
  ftpget.DoFtpTransfer();
}                                      // Close DoModal if block
```

Implement the DoFtpTransfer Function

There are five main steps to implement in the **DoFtpTransfer** function:

1. Create an Internet session.
2. Establish an FTP connection.
3. Add some initialization code for the string arrays that hold path information and set the type of copy operation desired.
4. Do the file transfer.
5. Close the FTP connection and Internet session.

➢ **Create an Internet session**

1. Use the Developer Studio editor to open Ftpget.Cpp.
2. Within the try block of the **CFtpGet::DoFtpTransfer** function, add the code to create an Internet session.

```
//
// 1) Create an Internet session
//
TRACE("creating session...\n");
CInternetSession session;
```

➢ **Establish an FTP connection**

Next, add the code to establish an FTP connection. Ensure that you check to see whether the connection is made through a proxy server and then uses the correct **GetFtpConnection** function call for each case.

```
//
// 2) Establish FTP connection
//
TRACE("getting ftp connection...\n");
// Heuristic:  If no gateway is specified then use server name directly,
//  otherwise use gateway as proxy server.
// See your network admin for your proxy name, or use an internal site.
//
if (m_strGateway.IsEmpty())
{
    m_ftp= session.GetFtpConnection(m_strServer);
}
else
{
    m_ftp= session.GetFtpConnection(
        m_strGateway,
        "anonymous@" + m_strServer,
        "YourName@YourCom.com"
        );
}
TRACE("Connection made\n");
```

➢ **Initialize path information and set the type of copy operation**

Add the following code to initialize the string arrays and set the type of copy operation to be performed.

```
//
// 3) Some miscellaneous initialization
//
// If no server path specified, make a "wildcard path".
if (m_strPath.IsEmpty())
{
    // If the path string is empty make a "wildcard path" to catch
// everything (up to number of path elements). Also, turn off "copy leaf
files only"
// since we don't have any knowledge of the leafs at this time.
    m_strPath= "*\\*\\*\\*\\*\\*\\*\\*\\*\\*\\";
    if (m_CopyType==FC_LEAFFILES)
    {
        m_CopyType= FC_ALL;
    }
}
// Split path string up and place into string arrays, one for each path
element.
PathToArray(m_strPath, m_arrStaticPath);// original path as user entered.
PathToArray(m_strPath, m_arrDynamicPath);    // path with wildcard
characters
                // replaced with matched directories.
```

➢ **Invoke the function to do the file transfer**

Call the member function to do the remote directory traversal and file copying. In the next subsection of this exercise, you will implement the **CFtpGet::CopyDirectory** function.

```
//
// 4) Begin file transfer starting at root of FTP server.
//
CopyDirectory(0);
//
```

➢ **Close the FTP connection and Internet session.**

Add code to close the connection and the session.

```
//
// 5) Close things up
//
m_ftp->Close();
delete m_ftp;
session.Close();
```

Implement a Recursive Search and File Retrieval Function

The function **CFtpGet::CopyDirectory** performs the search for desired file types and the retrieval, if that option is set. It also searches the directory for subdirectories that may be candidates for further searching and then recursively calls itself with the new directory level.

There are five main steps to implement in the **CopyDirectory** function:

1. Add initialization code to set the current directory on the FTP server to the current path level to be searched and to set a flag to indicate whether the directory should be copied.
2. Use the file type string to search for the desired file type(s) in this directory.
3. Determine whether the end of the path has been reached or to proceed down to another subdirectory and take the appropriate action in either case.
4. Check for subdirectories that match the desired path and store the names in a string array.
5. For each matching subdirectory found, set the dynamic path array and then recursively call **CFtpGet::CopyDirectory** with an updated path value.

➢ **Add initialization code**

1. Use the Developer Studio editor to open Ftpget.Cpp.
2. Add initialization code to set the current directory on the FTP server to the current path level to be searched and to set a flag to indicate whether the directory should be copied.

```
    //
    // 0) Initialization
    //
    // Set source directory at ftp site.
    // A level of 0 is the server root, 1 is first sub-dir, etc
    if (!SetFtpDirectory(PathLevel))
    {
        TRACE("Cannot set ftp directory\n");
        return FALSE;
    }

    // Set a flag to determine whether or not this is a dir to copy.
    BOOL bCopy= (m_CopyType==FC_ALL) ||
        ((m_CopyType==FC_LEAFFILES)&&(PathLevel ==
m_arrDynamicPath.GetSize()));
```

➢ **Search for the desired file type(s) in this directory**

Use the file type string to search for the desired file type(s) in this directory.

```
    //
    // 1) Find desired files for current directory and copy.
    //
             CFtpFileFind ff(m_ftp);
    BOOL success= ff.FindFile(m_strFileTypes);
    while (success==TRUE)
    {
        {
            // Need to call FindNextFile before doing any attribute
methods.
            success=ff.FindNextFile();

            CString strFilename= ff.GetFileName();
            if (!ff.IsDirectory())
            {
                // Get destination directory, create if needed.
                CString DestinationDir;
                if (GetLocalDirectory(PathLevel, DestinationDir))
                {
                    CString FullName= DestinationDir + strFilename;
                    // Do the copy.
                    if (bCopy) {
                        if (!m_ftp->GetFile(strFilename,FullName)) {
                            TRACE("Cannot put to local directory, error:
%d\n", ::GetLastError());
                        }
                    }
                    // Output file information.
                    static DWORD count= 0;
                    DWORD filesize= ff.GetLength();
                    m_TotalFileSize += filesize;
                    TRACE("%ld: %ld (%ld) %s\n", ++count, filesize,
m_TotalFileSize, FullName);
                }
                else
                {
                    TRACE("Cannot access local directory\n");
                    return FALSE;
                }
            }
        }
    }
    // Call Close to reset the search.
    ff.Close();
```

➢ **Determine appropriate action to pursue for search**

Add code to determine whether the end of the path has been reached or to proceed down to another subdirectory and take the appropriate action in either case.

1. If at the end of the path specification, then just return.

```
    //
    // 2) Terminate here if at end of path specification.
    //

    if (PathLevel >= m_arrDynamicPath.GetSize())
```

```
      {
          return TRUE;
      }
```

2. Otherwise, if there are no wildcard characters at the current path element, proceed to that subdirectory.

```
      else if (!strchr(m_arrStaticPath[PathLevel],'*') && !
  strchr(m_arrStaticPath[PathLevel],'?'))
      {

          // Just go down into path if no wildcards

          if (CopyDirectory(PathLevel+1)==FALSE)
              return FALSE;

      }
```

➢ **Store the names of subdirectories that match the current path element**

Otherwise, check for subdirectories that match the current path element and store the names in a string array.

```
        else
        {

        //
        // 3) Start a new search looking for sub-directory names only.
        //

            CStringArray arrSubDirNames;
            success= ff.FindFile(m_arrStaticPath[PathLevel]);
            while (success==TRUE)
            {
                {
                    // Need to call FindNextFile before doing any attribute
    methods.
                    success=ff.FindNextFile();
                    // Only concerned with sub-directories.
                    if (ff.IsDirectory())
                    {
                        CString strFilename= ff.GetFileName();
                        // Ignore special directory names, "." and "..".
                        if (strFilename!="." && strFilename!="..")
                        {
                            // Save matching sub-directory names.
                            arrSubDirNames.Add(strFilename);
                        }
                    }
                }
            }
            // Call Close to reset the search.
            ff.Close();
```

➢ **For each matching subdirectory, recursively call CFtpGet::CopyDirectory**

For each matching subdirectory found, set the dynamic path array and then recursively call **CFtpGet::CopyDirectory** with an updated path value.

```
      //
      // 4) Recursively copy matching sub-directories.
      //

          for (int idx= 0; idx< arrSubDirNames.GetSize(); idx++)
          {
              // Replace wildcard name with matching name.
              m_arrDynamicPath[PathLevel]= arrSubDirNames[idx];
              // Recursive call for new sub-directory, end copy if any errors
  are encountered.
              if (CopyDirectory(PathLevel+1) == FALSE)
                  return FALSE;
          }
      }   // end of else
```

➤ **Build and test your application**

If you are currently connected to the Internet, you can accept the default search parameters set in the baseline application to search the Microsoft FTP server for Knowledge Base articles. The completed code for this exercise is in \Labs\C11\Lab03\Ex01.