

By Edmond Woychowsky

Takeaway

While a relatively old technology, the File Transfer Protocol (FTP) is still a viable and useful communication method. This download explains how an FTP server can be implemented using the Microsoft .NET Framework and the C# programming language.

Table of Contents

FTP IN C#	2
SOCKETS.....	2
CALLBACKS	2
THE FTP COMMANDS	2
<i>Table A</i>	3
<i>Table B</i>	4
USING THE FTP CLIENT	4
<i>Table C</i>	5
AN EXAMPLE	5
<i>Listing A</i>	5
<i>Listing B</i>	5
STILL VIABLE	6
TECHNICAL INFORMATION.....	6
TECHREPUBLIC.....	7
<i>Additional resources</i>	7
<i>Version history</i>	7
<i>Tell us what you think</i>	7

FTP in C#

With the exception of perhaps the Amish and most elected officials, most of the general population has at least heard of [FTP](#) (File Transfer Protocol). Commonly used for copying files to and from other computers and dating back to the early 1970's most modern computers have some form of FTP utility hidden somewhere in their depths. Unfortunately, at least in the case of a Windows machines, invoking the included FTP utility requires use of the command prompt which can make things somewhat cumbersome when attempting to FTP from a program.

In fact, in order to use the Windows FTP utility from a program it is necessary to create a batch file containing the sequence of FTP commands on the local machine. This batch file would then be executed on the local machine with any files retrieved from the FTP server being written to the local machine. While this method would work, there are times when writing repeatability to the local machine would be undesirable, like say when using a Web application.

The only other alternative is to purchase an add-on to [Visual Studio .NET](#), which would provide a full implementation of an FTP client. However, my needs are simple and my budget is non-existent. Because of this, I decided to try an alternate approach using C# and sockets.

Sockets

Network programming, long considered a dark art, like setting the clock on a VCR or baking chocolate soufflés, has moved into the reach of ordinary mortals through the use of Microsoft's [.NET Framework](#). The Framework provides two namespaces, *System.Net* and *System.Net.Sockets*, which are instrumental in taking the hard work out of creating an FTP client.

The *System.Net* namespace contains the Domain Name Service (DNS) methods used to translate a URI into an IP address. This means that I type in the human-friendly [www.cnet.com](#) and get the network-friendly [216.239.115.141](#). In addition, this namespace provides the *IPAddress* and *EndPoint* classes, used to specify an IP address and logical port; while the *System.Net.Sockets* namespace provides the class necessary to create the actual socket.

When creating an FTP client there are two possible methods used to handle communications, synchronously or asynchronously, or as I usually think of them; wait until it is ready or call me when it is ready. Each has its advantages and disadvantages, for example, it is much easier to code a synchronous client and it is much easier to follow. An asynchronous client, while somewhat more difficult to follow, provides a timeout mechanism which keeps things moving along when things don't go quite as planned. Mostly, because of the possibility of timeouts I chose to code an asynchronous client.

Callbacks

In the .NET environment asynchronous callbacks are handled through the use of delegates, what would have been called a function pointer in the Pre-.NET world. There is, however, one potential problem when using asynchronous callbacks, race conditions. A race condition occurs when it is unpredictable which of two or more threads will finish first. Thankfully, it is easy to avoid a race condition for this application by simply having the main thread wait for the child thread to finish.

The FTP commands

On the World Wide Web Consortium's ([W3C](#)) Web site there is a little known document called [RFC-959](#) (Request For Comments). Dating back to October 1985, this document covers more about the inner workings of FTP than most people ever want to know. It covers the history of FTP and, more importantly, the thirty-three commands that comprise FTP which are listed in **Table A** along with those commands that I chose to implement. Another subject that RFC-959 covers is the various possible responses to the aforementioned commands, which are shown in **Table B**.

Table A

Command	Command Type	Implemented	Description
ACCT	Access control	Yes	Account
CDUP	Access control	Yes	Change to parent directory
CWD	Access control	Yes	Change working directory
PASS	Access control	Yes	Password
QUIT	Access control	Yes	Logout
REIN	Access control	Yes	Reinitialize
SMNT	Access control	No	Structure mount
USER	Access control	Yes	User name
ABOR	Service	Yes	Abort
ALLO	Service	No	Allocate
APPE	Service	Yes	Append with create
DELE	Service	Yes	Delete
HELP	Service	Yes	Help
LIST	Service	Yes	List files in current directory
MKD	Service	Yes	Make directory
NLST	Service	Yes	Named directory listing
NOOP	Service	Yes	No operation
PWD	Service	Yes	Print working directory
REST	Service	No	Restart
RETR	Service	Yes	Retrieve
RMD	Service	Yes	Remove directory
RNFR	Service	Yes	Rename from
RNTO	Service	Yes	Rename to
SITE	Service	Yes	Site parameters
STAT	Service	Yes	Status
STOR	Service	Yes	Store
STOU	Service	Yes	Store unqie
SYST	Service	Yes	System
MODE	Transfer	No	Transfer mode
PASV	Transfer	Yes	Passive mode
PORT	Transfer	No	Data port
STRU	Transfer	No	File structure
TYPE	Transfer	No	Representive type

Table B

Code	Description	Code	Description
110	Restart marker reply	350	Requested file action pending further information
120	Service ready in nnn minutes	421	Service not available, closing control connection
125	Data connection already open; transfer starting	425	Can't open data connection
150	File status ok; about to open data connection	426	Connection closing; transfer aborted
200	Command ok	450	Requested file action not taken
202	Command not implemented; superfluous at this site	451	Requested file action aborted; local error in processing
211	System status or system help reply	452	Requested action not taken
212	Directory status	500	Syntax error, command unrecognised
213	File status	501	Syntax error in parameters or arguments
214	Help message	502	Command not implemented
215	NAME system type	503	Bad sequence of commands
220	Service ready for new user	504	Command not implemented for that parameter
221	Service closing control connection	530	Not logged in
225	Data connection open; no transfer in progress	532	Need account for storing files
226	Closing data connection	550	Requested action aborted
227	Entering passive mode (h1,h2,h3,h4,p1,p2)	551	Requested action aborted; page type unknown
230	User logged in	552	Requested file action aborted; exceeded storage allocation
250	Requested file action ok, proceed	553	Requested action not taken; file name not allowed
257	"PATHNAME" created		
331	User name ok, need password		
332	Need account for login		

The way it works is that the FTP commands are sent to the server via port 21 and responses are returned via the same port. While this seems pretty simple, there is one curve; data isn't sent on port 21. The port used for data depends upon whether you need to use an active data port or a passive data port. With an active data port the client specifies the port (PORT command) and the server initiates the connection, while with a passive data port the server specifies the port (PASV command) and the client initiates the connection. Because my application resides behind a firewall, the option of using an active data port is out of the question for security reasons, only the PASV command has been implemented.

Using the FTP client

My implementation of an FTP client requires creating an instance of the object and applying the methods in the correct order. For example, before sending a RETR (retrieve) command it is obvious that a PASV (passive data port) command needs to be sent, because data is being retrieved from the FTP server. But it isn't quite so obvious that the same is also true for the LIST (list directory files) command. Because of this, **Table C** lists the commands that require a data port.

Table C

Command

LIST
APPE
NLST
RETR
STOR
STOU

An example

Now that the rules have been laid out let's walk through an actual example and step through the code shown in **Listing A**.

Listing A

```
FTPClient ftp = new FTPClient("127.0.0.1");

    ftp.syst();
    ftp.cwd("usr");
    ftp.pasv();
    ftp.list();
    ftp.quit();
```

The first line defines the FTP server, my local machine in this case, and the user and password. The second line, the `syst()` method requests the system type of the FTP server. The `cwd()` method changes the working directory to `/usr`. The next two methods `pasv()` and `list()` basic tell the FTP server to pick a port and list the files in the current directory. The final method, `quit()`, ends the interaction with the server. **Listing B** shows the results.

Listing B

```
Attempting connection to FTP server...
220 Microsoft FTP Service
USER anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
PASS *****
230 Anonymous user logged in.
SYST
215 Windows_NT
CWD usr
250 CWD command successful.
PASV
227 Entering Passive Mode (127,0,0,1,10,184).
LIST
125 Data connection already open; Transfer starting.
226 Transfer complete.
11-02-04 02:28PM          1646 asnFields.xml
06-27-03 04:24PM          972236 DeployingDotNetApps.pdf
01-20-05 02:59PM           6416 rfc1579.txt.pdf
01-20-05 02:57PM           85028 rfc959.txt.pdf
03-03-03 09:24AM        2147968 Team Development Guide.pdf
01-20-05 02:43PM        6472624 UML.pdf
QUIT
221
```

Still viable

At first glance, writing a FTP client in the early twenty-first century may seem a bit like investing heavily in a company that makes buggy whips. However, when the amount of data produced by legacy applications is considered it begins to make some sense. Although the days left to these applications may be numbered, they and their data are still around. Because of this, FTP is still a viable method of moving files around.

Technical information

The PROGRAM.ZIP file accompanying this PDF contains the C# source code and an example compiled executable of the FTP client described in the download. The source code and executable are provided as a learning tool for the sole purpose of expanding on the information presented in the article portion of this download. The source code and executable are not intended to be used as a fully-functional application. It is highly recommended that the PROGRAM.ZIP file be extracted to a development environment and not to a production machine connected to an enterprise network. Consult your organization's policies on software installation.

TechRepublic

Additional resources

TechRepublic's free Software/Web Development NetNote newsletter highlights key topics, including XML, Java, .NET, SQL Server, and Web services. [Automatically sign up today!](#)

- [Transferring Files With GridFTP](#) (White paper)
- [Create single instance objects in your .NET applications to improve performance](#) (Download)
- [Programming quality spreadsheet](#) (Download)

Version history

Version: 1

Published: 3/8/2005

Tell us what you think

TechRepublic downloads are designed to help you get your job done as painlessly and effectively as possible. Because we're continually looking for ways to improve the usefulness of these tools, we need your feedback. Please take a minute to [drop us a line](#) and tell us how well this download worked for you and offer your suggestions for improvement.

Thanks!

—The TechRepublic Downloads Team