Programming Guide

Revision 5.1.2

4705 Crystal Lake Dr
Hilliard, OH 43026
USA

3 February 2013

Version History

Date	Version	Modifications
2013-2-3	5.1.2	*Bug Fixes:* Modules now install in REAL Studio projects properly.
2009-6-16	5.1.1	*Added:* FTPSessionClass.GetFTPSuiteVersion returns version string. *Bug Fixes:* FTPSessionClass.PutSingleFile and FTPSessionClass.StartPutFile now work properly in the Active mode.
2008-6-29	5.1.0	*Added:* Support for Linux. Tested with several different Linux distributions *Bug Fixes:* Delete Directory Smart Command works properly with file names that contain spaces. *Depreciated:* REALbasic Standard Edition no longer supported.
2007-10-3	5.0.0	*Added:* Support for FTP over SSL (FTPS). Better support for files and folders with dates as names.
2007-8-9	4.2.3	*Bug Fixes:* No longer hangs up when connection is lost.
2006-9-12	4.2.2	*Bug Fixes:* Manual typo on page 20: Mode default was listed as Active; it is Passive. *RestartPutSingleFile* and *RestartGetSingleFile* now work properly. Under some conditions, a file was stored or retrieved in its entirety instead of from the point where the transfer was interrupted. The FTP transfer timeout timer is now working properly for all of the Smart Commands. Using GetSingleFile, StartGetSingleFile, GetDirectory and StartGetDirectory in a thread now works properly. It was possible for the 226 response from RETR to occur *before* the final 102 status is received by the data socket. That meant that the thread would start the next command (if there is one) before the data socket has a chance to close the binary stream, contaminating the downloaded file. *DeleteDir* now operates more like the other Smart Commands in that it does not change directory, but simply deletes the current directory on the FTP server. *StartDeleteDir* no longer causes an "OutofBounds" exception if FTPSessionClass.SetServerDirectoryPathName is passed a top directory without putting in the initial "/" in the path as one is automatically placed there.
2005-12-14	4.2.1	*Bug Fixes: StartGetDir* and *GetDir* no longer corrupt the first file transferred by appending a directory list to the file.
2005-11-1	4.2	*Added:* SITE Command, Direct Command *Bug Fixes: StartGetDirList* and *GetDirList* no longer cause a stack overflow when used with some FTP servers.
2005-04-20	4.1	*Added:* Abort Transfer, Restart Get Single File and Restart Put Single File Smart Commands. *Bug Fixes:* ParseDirListLine now properly parses file and folder name that contain spaces on the standard FTP Server included with Windows.
2005-01-30	4.0	*Added:* New Architecture, Multiple session support, speed increase, SetLocalFolderItem, ParseDirListLine, ParentDir Smart Command, New Examples *Bug Fixes:* FTP Server messages are now correctly reported by Response_FTP event instead of Status_FTP event, which now only reports FTP Suite-generated messages. A child folder with the same name as a parent folder no longer causes an error.
2004-09-15	3.4	*Added:* Set/GetCommandPort methods to FTPSessionClass *Bug Fix:* Missing FTP Suite User Messages for Successful_FTP event added.
2004-07-08	3.3	*Added:* Directory Listings now display invisible files and directories. CancelFTPSession to replace EndFTPSession which does not disconnect when using partial sequences. *Bug Fix:* Large (> 8-10Mb) files are no longer truncated when downloaded. Large (>40-50 lines) directory listings are no longer truncated. NOTE: REALbasic 4.5 (and earlier) is no longer supported due to increased reliance on 5.x features.
2004-04-19	3.2	*Added:* GetDir, PutDir, DeleteDir. Support for REALbasic 4.5, Communications from the FTP server can be converted to and from the server's native text encoding.

2004-02-23	3.1	*Added:* RemoveDir, MakeDir, ChmodSingleFile. GetWorkingDirectory, Errors during FTP Commands no long disconnect client from FTP Server. *Bug Fix:* StartPutDir is now working. A bug was introduced in the 3.0 release. Corrected minor documentation error.
2003-12-15	3.0	*Added:* Status Events, Localizable FTP Suite Messages, Variable Send Buffer Size, Fine Transfer Sequence Control *Bug Fix:* Time out value does not have to be explicitly set. It defaults to 20 seconds as de documentation. *Removed:* Status GUI Binding, Support for RB.4x and earlier. FTP Suite no longer supports the original REALbasic Socket Control and must be used with REALbasic 5.x.
2003-07-22	2.2	*Added:* Set Timeout, Name List option for GetDirList
2003-06-30	2.1	*Added:* RenameFile
2003-06-15	2.0	*Added:* Set Passive Mode (PASV), MuliLine User Status, Uses REALbasic 5.x. *Bug Fixes:* GetDir no longer fails if a directory only contains sub-directories, PutDir ignores files that have no length, DeleteDir no longer fails randomly.
2003-02-19	1.2	*Added:* User Defined Data Ports, Manual describes use of data ports fully. *Bug Fix:* FTPSuiteSerialNumber is back to being a global, Documentation is more accurate.
2003-02-06	1.1.2	*Bug Fix:* Number of global declarations dramatically reduced, Setting status field parameters to NIL no longer causes crash.
2003-01-03	1.1.1	*Bug Fix:* Put Directory now works. (Bug introduced in 1.1 release)
2002-12-16	1.1	*Added:* PutDirectory, EndFTPSession, GetFileList, Configurable transfer type and FTPTransferDone flag. *Bugs Fixes:* Transferring directories to NT servers no longer generates FTP errors.
2002-11-06	1.0	First public release

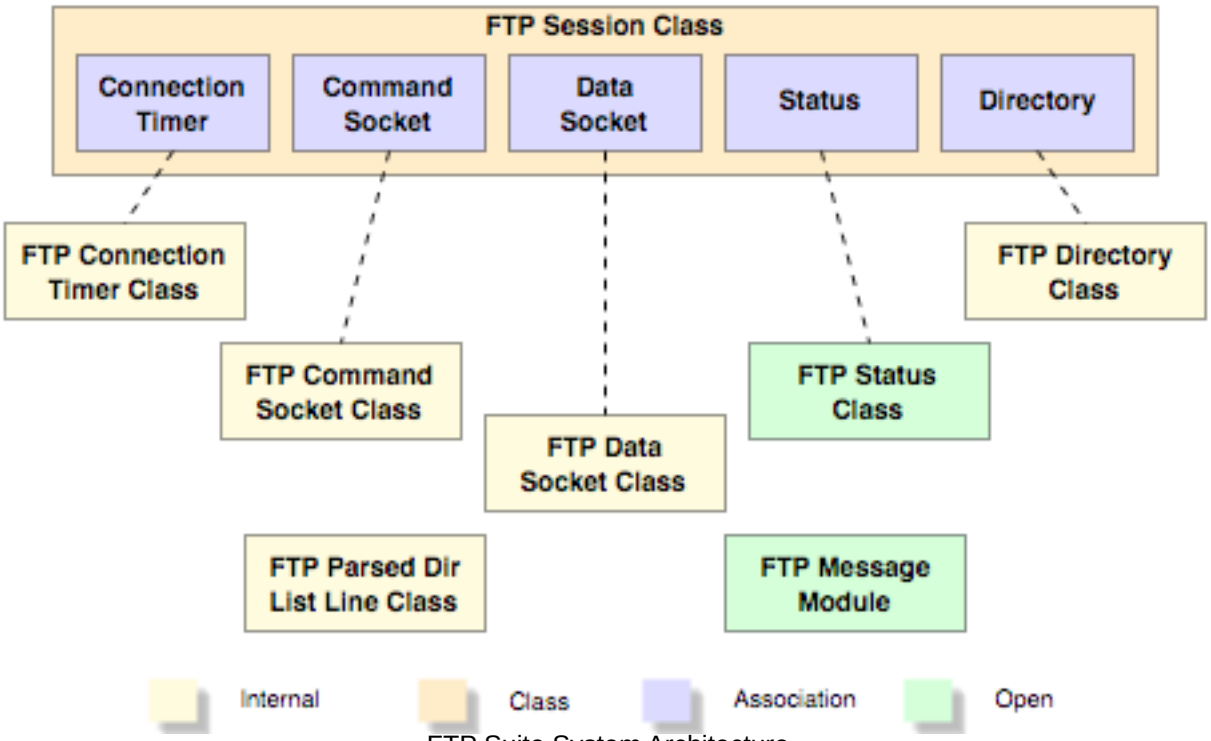
Overview

FTP Suite is a collection of classes and code modules that allows REALbasic applications to implement the FTP protocol. There are two types of FTP operations you can perform with FTP Suite.

Smart Sequences perform an entire FTP sequence from login to logoff. They are suited for applications where transfers occur mostly under the client program's control and for transfer sequences that affect entire directories or multiple files in the same directory.

Smart Commands perform partial FTP sequences (usually one or two FTP commands) and are best suited for clients that cannot predict what FTP actions will be needed before hand or clients who need transfer files beyond the standard folder to folder transfer. Smart Commands give the developer more flexibility.

The FTP Suite architecture is composed of eight software components and is outlined on the figure and table below.



FTP Suite System Architecture

Component	Purpose	Access	Open
FTP Session Class	Contains all developer accessible methods and storage to setup and execute FTP transfers.	User	No
FTP Status Class	Provides accessible events to allow developer to determine status of an FTP transfer.	User	Yes
FTP Message Module	Provides access to FTP Suite-generated messages to allow modification and multiple-language support.	User	Yes
FTP Command Socket	Sends FTP commands to FTP server and processes responses.	Internal	No
FTP Data Socket	Sends and receives all data between client and FTP server.	Internal	No
FTP Connection Timer	Raises Error Event if not reset within timeout period.	Internal	No
FTP Parsed Dir List Line Class	Provides a structure for a parsed line of directory listing data.	Internal	No
Directory Class	Provides a structure for the local directory tree.	Internal	No

Installation and Setup

FTP Suite is installed in your REALbasic project by simply dragging the entire FTP Suite folder into REALbasic project window. After installation, all required methods are available to the REALbasic project.

The only requirement for setting up FTP Suite is to set the serial number variable to the serial number you received when you registered the product. Add this line (using your own serial number) to the Open event of your application:

```
FTPSuiteSerialNumber= "FTPSUITE50-0000000-000000000"
```

Note: FTP Suite is only compatible with REALbasic Professional 2006 Release 3 and later.

Usage Guidelines

The FTP Session Class is the main class that the developer will work with on a regular basis. Within it are all the methods, properties and class instances required for a single FTP Session. Multiple instances of this class allow multiple FTP session streams.

Construction of an FTP Session Class instance automatically creates an instance of the FTP Data and Command Sockets, Connection Timer Class, Directory Class and Status Class. The FTP Message Module is shared by all instances of FTP Session Class.

To set up an FTP session, an instance of the FTPSessionClass must be created in dedicated property. In the case of an FTP Sequence this property can be transient if there will be no other FTP communication to the same server. If several FTP Smart Commands are to be executed, the property needs to remain accessible to the controlling REALbasic code for the duration of the session. It may also to do this with Smart Sequences for the same reason.

To enable FTP with SSL/TLS encryption, set FTPSessionClass,FTPS to true. Make sure the FTPCommandSocket's super is set to SSLSocket. The FTP Server must be set to handle SSL and have a valid certificate.

Smart Sequence Setup

At the minimum, a Smart Sequence needs to know how to login to the ftp server and what directory it will be working with on the ftp server. As an example, the setup to get a full directory listing from a directory off the root directory called "stuff" would consist of the following:

```
Dim ftp_session As New FTPSessionClass

// set FTP login info and server directory
ftp_session.SetLoginInfo("ftp.your.domiain", "user id", "password")
ftp_session.SetServerDirectoryPathName("/stuff")

// start the smart sequence
ftp_session.StartGetDirList
```

Since the FTP configuration properties are set to the most common settings by default, this is all the setup usually required.

To further expand the example, let’s say the application scanned the returned directory and needs to send transfer a directory that is missing from the client. If the property ftp_session is still available, the login information can be re-used.

Using the same FTPSessionClass instance, we just need to reset the server directory path to include the missing directory, named "new" in this case. The location of the client directory that will be transferred to the server is needed as well.

```
ftp_session.SetServerDirectoryPathName("/stuff/new")

// get folder item for client source directory and set for ftp session
Dim f As FolderItem
f = ApplicationSupportFolder.Child("app name").Child("client new")
ftp_session.SetLocalDirectory(f)

// start the smart sequence
ftp_session.StartPutDir
```

Note that this is not the most efficient method, since a Smart Sequence logons on, changes to the server directory path, and logs off *every* time. However, for occasional multiple sequences, the convenience may be worth the small overhead in the FTP sequence. See Smart Command Usage section for additional information.

Renaming Files

This section will describe the process of renaming file(s) to a host machine. First, a new instance of FTPSessionClass is created and all the appropriate login and server path information is set.

```
// demo on how to rename a remote file
Dim Session as FTPSessionClass

// new transfer class holds all info
Session = New FTPSessionClass
// set info for login
Session.SetLoginInfo(ServerFld.text, UserIdFld.text, PassWordFld.text)
// set server directory path
Session.SetServerDirectoryPathName(ServerDirFld.text)
```

New and old file names are now added. Files names must always be added in pairs: New name, *then* Old name. FTP Suite will issue an error if an odd number of file names are present, but that is the extent of the setup verification. The files must be in the same directory. Renaming of directories is not supported.

```
// add new and old server file names
Session.AddFileName "FileOne"// First new name
Session.AddFileName "File1"     // First old name
Session.AddFileName "FileTwo"// Second new name
Session.AddFileName "File2"     // Second old name
```

All that is left is to set the transfer type and user status options and execute StartRenameFile

```
// set transfer type - I for binary, A for ASCII
Session.FTPTransferType = "I"
// set status reporting
Session.SetStatusReporting(True)

// start renaming process
Session.StartRenameFile
```

Handling Multiple Sessions

Applications having the requirement of needing to support multiple, simultaneous FTP sessions, can create multiple instances of the FTPSessionClass; one for each session. Since there is no way to know how many sessions will be running at any time, a global, dynamic array of FTPSessionClass instances should be created:

```
Session(-1) As FTPSessionClass
```

FTPSessionClass has a string property, ID, which allows developers to save a unique ID with each session. It could be a file name or simply an incrementing number. This is done by providing a parameter string when the FTPSessionClass is instantiated.

A “droplet” application that sends files to a fixed directory when you drop them on the application’s window is an example of using multiple sessions. There is no way to know how many files will be dropped on the application, so we must generate a new session for every file dropped. The application’s DropObject event is where this is done:

```
Do
    f = Obj.FolderItem
    If Not f.Directory Then // as long as it’s a file

        // create new session and ID using upper bound of session array
        id = Ubound(Session)
        session.Append New FTPSessionClass(Str(id))

        // use id to access FTPSessionClass instance and set up transfer
        session(id).SetLoginInfo(Server,LoginID,Password)
        session(id).SetServerDirectoryPathName(ServerPath)
        session(id).SetLocalFolderItem(f)
        session(id).StartPutFile

    End If
Loop Until Not obj.NextItem
```

Smart Command Usage

One of the reasons to use Smart Commands is to make large sequential transfers more efficient by logging on and off only once and changing directory only when needed. When using Smart Commands, the must provide a way for the code to be able to wait until each command is complete before issuing the next one. A good way to do this is to provide a global Boolean parameter, "FTPDone" which is set in the Successful_FTP event of the FTPStatusClass. (For more detail, see *Handling Status* section.) It's also a good idea to handle errors the same way: a global Boolean, "FTPError" set in the Error_FTP event of FTPStatusClass. Each time a call to a Smart Command is set up, both Booleans are set to false.

Because the code will loop testing for FTPDone, the entire transfer code needs to be placed in a thread or a timer. Otherwise, user interface elements will be frozen until the transfer is complete. In most cases, a thread is more appropriate because it tries to maximize its CPU usage and that trait will speed up the transfer.

If the "droplet" application, discussed above, must transfer a large number (30+) of files to the same directory with each "drop", the efficiency of the transfers would be increased if Smart Commands were used instead of Smart Sequences. First, the "DropObject" event will simply create an instance of FTPThreadClass, which will hold all the code to execute the transfer. The DragObject will be passed to FTPThreadClass and the thread's Run method executed.

In FTPThreadClass, there are local methods to logon, logoff, change directory and transfer the file. Below is the FTPLogin method:

```
Private Function FTPLogon() As Boolean
    // create new session and setup logon information
    Session = New FTPSessionClass
    Session.SetLoginInfo(Server,LoginID,Password)
    Session.SetStatusReporting(True)

    // reset flags and execute logon Smart Command
    FTPDone = False
    FTPError = False
    Session.Logon

    // wait for FTPDone and return the inverse of FTPError,
    // so true means no error
    Do
    Loop Until FTPDone = True
    Return(NOT FTPError)
End Function
```

The FTPChangeDir method is similar, but an instance of Session is not created, since the client should already be logged on:

```
Private Function FTPChangeDir(path As String) As Boolean
    // set the new path, relative from the current server dir.
    Session.SetServerDirectoryPathName(path)

    // reset flags and execute Changdir Smart Command
    FTPDone = False
    FTPError = False
    Session.ChangeDir
```

```
    // wait for FTPDone and return the inverse of FTPError,
    // so true means no error
    Do
    Loop Until FTPDone = True
    Return(NOT FTPError)
End Function
```

Finally, the FTPLogoff method simply executes Logoff. Since most FTP servers will logoff automatically even if the logoff command is not received, testing is not really necessary except for more stringent security requirements.

```
Private Function FTPLogoff()
    // reset flags and execute Logoff Smart Command
    FTPDone = False
    FTPError = False
    Session.Logoff

    Return
End Function
```

The Run event of FTPThreadClass uses these three private methods:

```
Sub Run()
    Dim f As FolderItem

    // logon and change to server folder
  If Self.FTPLogon = False Then
    Return
  End If

  If Self.FTPChangeDir("/stuff/new")= False Then
    Return
  End If

    Do
        // using local copy of Obj, set by constructor
        f = Self.Obj.FolderItem
        If Not f.Directory Then // as long as it's a file

            // use current session to set up single file transfer
            Self.Session.SetLocalFolderItem(f)
            Self.Session.PutSingleFile // Smart Command

        End If
    Loop Until Not obj.NextItem

    // logoff when done
    Self.FTPLogon
End Sub
```

Handling Status

Status events are generated by the FTPStatusClass, which is open for developer modification. This class can be utilized to provide user status messages and control logic flow within a REALbasic application. Status handling is normally disabled and is enabled by calling FTPSessionClass.SetStatusHandling with a parameter of True. Doing this creates an instance of FTPStatusClass, which is bound to the FTPSessionClass instance. The developer can then place code inside the appropriate FTPStatusClass events for user interface updates and control flow purposes. When multiple sessions are created, the Session property in FTPStatusClass holds an instance of the FTPSessionClass. This can be used to identify the session by accessing the ID property in FTPSessionClass:

```
Sub Successful_FTP(message As String)
    // update main window status field
    MainWindow.UpdateStatus(Self.Session.ID,message)
End Sub
```

The message parameter returned in Successful_FTP (and other FTPSessionClass events) can be compared against FTP Suite-generated messages so special messages can be handled differently. (See FTP Suite Messages section for more detail on these messages.) An example of trapping out a specific message is when the application needs to display a directory list that is sent back from the FTP server by a Smart Sequence or Command request. The following code opens a new window and populates it with the directory listing when the message parameter is equal to the FTPSuite-generated message: FTP_FileListComplete.

```
Sub Successful_FTP(message As String)
    Dim list As String

    // open directory window and load
    // if Successful_FTP due to directory request

    If message = FTP_FileListComplete Then
        flwin = New FileListWindow

        // save local copy of directory listing, replacing replace server's
        // line endings with client's for cross platform safety
        list = ReplaceLineEndings(Session.DirectoryFileList,EndOfLine)

        // put file list in editfield
        flwin.FileListEditField.Text = list

    Else
        // update main window status field for other messages
        MainWindow.UpdateStatus(Self.Session.ID,list)
  End If
End Sub
```

See FTPStatusClass in the FTP Suite API Reference section for more information.

Parsing Directory Listings

Developers can make use of a Smart Command, ParseDirListLine to extract information from one line of a full directory listing returned from an FTP server. A most directory formats are handled properly. ParseDirListLine takes a line of text as a parameter and returns an instance of the FTPParsedDirListLineClass. This class has the following properties:

Name	String	This property contains the file or folder name.
Size	Integer	This property contains the file size in bytes.
ModDate	String	This property contains the file or folder's modification date in the native listing format.
ModTime	String	This property contains the file or folder's modification time in the native listing format. If there was no modification time found, "12:00" is placed in the property.
Permissions	String	This property contains the file or folder permissions in "rwxrwxrwx" format.
IsDir	Boolean	This property is true if the item is a folder or directory.
IsFile	Boolean	This property is true if the item is a file. Note: IsDir and IsFile are mutually exclusive; they cannot both be true for the same item. The can, however, both be *false* if it is not clear what the item is.

Text Encoding

With the release of REALbasic 5.0, developers were given more options in handling multiple text encodings. This means that, especially when using accented characters, you need to explicitly specify what text encoding a server is using. There are a few default encodings to try depending on the OS that the FTP server is running on:

UNIX ISOLatin1 *Windows* WindowsLatin1 *Mac OS* MacRoman

However, it's very possible that other encodings will be used. The default encoding for FTP Suite is ISOLatin1. If you find that this does not work, you can change it using FTPSessionClass.SetServerEncoding.

Data Port Information

The default FTP data port is 20. The works for single file transfers *only*. The reason for this is the *stream* transfer mode is the default FTP transfer mode, which requires that the end of the file be defined by closing the data port connection.

This causes a problem because FTP Suite is designed to transfer multiple files in a single session and the TCP protocol is required to hold the connection record for a time out period to guarantee the reliable communication. This means the same data port cannot be re-opened immediately. Trying to will return an error from the FTP server indicating the data port is busy.

To allow FTP Suite to transfer a new file immediately after finishing another, it uses a pre-defined range of data ports, sequencing through them as each file is transferred. Additionally the initial data port of an FTP transfer session is randomly selected from the data port range, which reduces the risk that closely spaced FTP sessions will attempt to use the same data port.

The default range of data ports is 30003 to 30100. This range is used if SetDataPortRange is not called to set up a custom data port range.

A list of assigned data ports can be found on the IANA home page at: http://www.iana.org/assignments/port-numbers.

The requirement for setting a data port range is eliminated when the Passive mode is used. In the Passive mode, the *server* returns a data port to transfer data over. Not only is this more convenient, but it is inherently safer from a security point of view.

Example Applications

There are five REALbasic projects included with FTP Suite. One, FTP Suite Test Rig.rb, is what Apogee Development, Ltd. utilizes to test all methods and properties. While not a typical example application, it can be useful to examine its code and for a way of trying out various Smart Sequences, Smart Commands and FTP Suite properties.

Upload Droplet.rb and Upload Droplet Multiple.rb are the same application: it uploads any file that is dropped on the application's window to a fixed directory on an FTP server. The "multiple" version can handle multiple files; the other version transfers the first file it encounters, ignoring the rest. These applications are a good example of using a Smart Sequence, handling status, and multiple transfers.

Folder Info is another "droplet" application with a special task: it recursively parses through a remote folder and counts the number of folders, files and the number of bytes used. It's a good example of using Smart Commands, recursive and thread-based operations, and using the FTPSessionClass.ParseDirListLine command.

Finally, Server Browser.rb implements a simple EditField-based application that allows the user to logon to a remote FTP server and browse its contents. When a file or folder is selected, its permissions can be changed in separate window. This application was specifically written to demonstrate the use of the FTPSessionClass.ParseDirListLine and FTPSessionClass.ChmodSingleFile Smart Commands.

NOTE: These sample applications are supplied to illustrate methods of the use of FTP Suite. While the examples are free to use, they have not been fully tested and are not guaranteed. Apogee Development, Ltd. assumes no responsibility for issues related to the use of this code.

FTP Suite API Reference

This section discusses all the methods, properties and constants available to the FTP Suite developer.

FTPSessionClass

This class is the heart of all FTP Suite operations. All Smart Sequences and Smart Commands are contained in this class. It can be instantiated as needed to handle multiple FTP sessions.

Smart Sequence Methods

Note: If an error is detected during the execution of a Smart Sequence, the client is disconnected from the server.

StartDeleteDir
Deletes all the files and directories from the specified server directory.

StartDeleteFile
Deletes any number of files on the server. All files to be deleted must reside in the same server directory.

StartGetDir
Transfers all the content from a server directory and copies the entire directory structure to the local machine.

StartGetDirList
Transfers a list of all the files and subdirectories in a given server directory, storing it in the DirectoryFileList property contained in the FTPSessionClass. Optionally, a name list can be returned instead of the full detailed listing. Directory lists can display invisible files and directories by setting FTPSessionClass.ShowInvisibles to True.

StartGetFile
Transfers any number of files from the server to the local computer. All files must reside in the same server directory and will all be placed in the same local directory.

StartPutDir
Transfers all the content from a local directory and copies the entire directory structure to the server.

StartPutFile
Transfers any number of files from the local computer to the server. All files must reside in the same server directory and will all be placed in the same local directory.

StartRenameFile
Renames a file or list of files on the remote server. All files must reside in the same server directory.

Smart Command Methods

These commands execute part of an FTP sequence. The partial sequence may have only one FTP command or several depending on the complexity of the step. The client *must* be connected to the FTP server using the Logon method before any other partial sequence method can be used. Note that if an error occurs, the client will remain connected to the FTP host. This allows applications to handle such situations as attempts to remove non-empty directories and transferring non-existing files, for example.

AbortTransfer
Stops the current FTP operation without disconnecting or ending the session, allowing further FTP communication without re-connecting to the FTP server.

CancelFTPSession(UserMsg as String)
This method will stop an FTP full or partial sequence via program control. If UserMsg is supplied it will be present in the FTPStatusClass.Successful_FTP event's message parameter.

ChangeDir
Changes to the remote directory defined in FTPSessionClass.

ChmodSingleFile
Changes the permissions of a single file in the current directory on the FTP server. This command uses SITE command to execute a CHMOD command on the server. NOTE: It is **not** mandatory for FTP servers to support the CHMOD command, so error checking is important. The ChmodMask must be set in the usual UNIX style:

Mask**Attribute**400read by owner040read by group004read by anybody (other)200write by owner020write by group002write by anybody100execute by owner010execute by group001execute by anybody

DeleteDir
Deletes all the files and directories from the specified server directory.

DeleteSingleFile
Deletes a single file in the current directory on the FTP server.

DirectCommand(command As String)
Sends the command directly to FTP server. If the code returned from the FTP server is within 200-299, an FTP_Successful event is trigged. NOTE: this command's performance is highly server-specific. Commands that work on one brand of FTP server will probably not work for another brand. Check your FTP server documentation.

GetDir
Transfers an entire directory structure from the current directory on the FTP server to the local computer.

GetDirList
Returns a list of all the files and subdirectories in a given server directory, storing it in a property contained in the FTPSessionClass.DirectoryFileList. Optionally, a name list can be returned instead of the full detailed listing. Directory lists now display invisible files and directories.

GetSingleFile
Transfers a single file from the current directory on the FTP server to the local computer.

GetStatus
Returns system-specific status that can be any format. Note: this information is passed as a message in the Status_FTP event, so SetStatusReporting must be set to true. The status will start and end with a 211 code. Some servers do not support this command and return an error code of 500.

GetSystemInfo
Returns the type of operating system at the server. Note: this information is passed as a message in the Status_FTP event, so SetStatusReporting must be set to true. Its returns the number 215 followed by a single word, which is the system name.

GetWorkingDir
Returns the full path of the current working directory on the FTP Server. Note: this information is passed as a message in the Status_FTP event, so SetStatusReporting must be set to true. The path is enclosed within double quotes.

Logoff
Logs off the FTP Server.

Logon
Logs on to the FTP Server.

MakeDir
Creates a remote directory defined in FTPSessionClass of the FTP server.

ParentDir
Changes directly to the current working directory's parent directory.

ParseDirListLine(line As String) As FTPParsedDirListLineClass
Returns the full path of the current working directory on the FTP Server. Note: this information is passed as a message in the Status_FTP event, so SetStatusReporting must be set to true. The path is enclosed within double quotes.

PutDir
Transfers an entire directory structure from a local directory to the current directory on the FTP server.

PutSingleFile
Transfers a single file from a local directory to the current directory on the FTP server.

RemoveDir
Removes the remote directory defined in FTPSessionClass. NOTE: the directory must be empty.

RenameSingleFile
Renames a single file in the current directory on the FTP server.

RestartGetSingleFile
Continues a previously interrupted download of a single file.

RestartPutSingleFile
Continues a previously interrupted upload of a single file.

SITECommand(command As String)
Sends the command to FTP server using the FTP SITE command. NOTE: It is **not** mandatory for FTP servers to support the SITE command. The actual commands that are available via SITE vary, so check your FTP server documentation.

Set Methods

These FTPSessionClass methods are used to set up properties require special handling or as alternatives to setting them directly.

AddFileName(FileName As String)
Adds the FileName the File List.

DeleteFileName(FileName As String)
Deletes the FileName from the File List.

SetDataPortRange(Data Port Start As Integer, Data Port End As Integer)
Defines the range of data ports to use (see Data Port Information below) and sets the initial data port to randomly within the range specified. If SetDataPortRange is not called, the default data port range (30003 to 30100) will be used.

SetLocalDirectoryPathName(LocalDirectoryPath As String)
Defines the directory path on the local computer. If blank, FTP Suite will transfer to the directory where the application is located. If the User ID does not have permission to access this directory, then the FTP sequence will generate an error.

SetLoginInfo(ServerAddress As String, UserID As String, Password As String)
Sets up the initial login information required for a full FTP command sequence.

SetServerDirectoryPathName(ServerDirectoryPath As String)
Defines the Server directory path. If blank, FTP Suite attempts to use the server's root directory. If the User ID does not have permission to access root, then the FTP sequence will generate an error.

SetStatusReporting(EnableStatusEvents As Boolean)
When EnableStatusEvents is set to true, an instance of FTPStatusClass is created and FTP Suite will generate the status events described in the User Feedback section above. The default setting is False (no status events generated).

SetTimeoutSeconds(TimeoutSeconds As Integer)
Defines when a transfer operation will time out (in seconds). The default setting is 20 seconds.

Properties

These properties are can be accessed directly. Some also can be set using methods. (See Set Methods section.)

ChmodMask As String
Defines the mask that will be used by the CHMOD command to set the permissions of a file.

CommandPort As Integer
Defines the command port to use. Useful when FTP servers are specially configured for security reasons. The default Command Port is 21.

DataPort As Integer
Defines the data port to use (See Data Port Information section.). While this property can be set manually, it is usually set with FTPSessionClass. SetDataPortRange.

DirectoryFileList As String
After a successful StartGetDirList or GetDirList request, the directory list will be in this string.

FTPS As Boolean
When set, the session will be logged in under SSL. **Note, that FTPCommandSocket *must* have SSLSocket as its super and that the FTP server must be set up for SSL with a valid certificate.**

FTPTransferType As String
Defines the type code (“I” for binary, “A” for ASCII). The default setting is "I" (binary).

ID As String
Used to identify an FTPSessionClass instance, or session.

PassiveMode As Boolean
When set, all FTP operations will take place with the server in the passive mode. The default setting is true (Passive Mode Enabled).

SendBufferSize As Integer
Defines the size of the send buffer. Using large buffer sizes for known reliable connections can speed up transfers from client to server. Currently, there is no way to change the size of the receive buffer in REALbasic's CoreSocket class. The default size of the send buffer is 2Kb.

ServerAddressName As String
The domain name or IP address of the FTP Server. This may also be set with FTPSessionClass.SetLoginInfo.

ServerPassword As String
The password for the FTP Server account to be logged onto. This may also be set with FTPSessionClass.SetLoginInfo.

ServerTextEncoding As TextEncoding
Changes the server's Text Encoding to a REALbasic Text Encoding Class of the user's choice. The default setting is ISOLatin1 (the usual for UNIX systems). See Text Encoding Section.

ServerUserID As String
The User ID for the FTP Server account to be logged onto. This may also be set with FTPSessionClass.SetLoginInfo.

ShowInvisible As Boolean
When set, invisible files and directories will be returned in the directory list for both name and default lists. This feature uses the "-a" option (NLST -a and LIST -a). The default setting is False (hide invisible files/directories).

Timeout As Integer
Defines when, in milliseconds, an FTP operation will time out. The default setting is 2000 milliseconds. May also be set by FTPSessionClass.SetTimeoutSeconds.

UseNameList As Boolean
When set, a Name List (NLST) will be returned. The default setting is False (full list using LIST).
.

FTPStatusClass

This class supports the execution of status events and is accessible to the developer. This class can be utilized to provide user status messages and control logic flow within a REALbasic application.

Events

Error_FTP(message As String)
This event is triggered when FTP Suite detects an error. The message is generated by FTP Suite and can be localized by adding additional languages to the constants contained in the FTPMessages module. FTP Errors will only disconnect from the server when using FTP Sequences.

Response_FTP(message As String)
This event is triggered when FTP Suite receives a response from a command sent to the FTP server. The message is either generated by the FTP server, or is a command that FTP Suite is sending to the server. These cannot be localized by FTP Suite.

Status_FTP(message As String)
This event is triggered when FTP Suite sends a command to the FTP Server. The message is generated by FTP Suite and can be localized by adding additional languages to the constants contained in the FTPMessages module.

Successful_FTP(message As String)
This event is triggered when FTP Suite has successfully completed an FTP sequence with the FTP server. The message is generated by FTP Suite and can be localized by adding additional languages to the constants contained in the FTPMessages module.

Progress_FTP(bytecount As Integer)
This event is triggered when FTP Suite receives data from the FTP server. The message is current number of bytes transferred so far. Note that is a good idea to use this function as refreshing the display for every buffer transfer can significantly slow down the transfer.

The granularity of these messages is such that you can display and respond to different messages in different ways, such as only responding to errors, successful transfers and displaying bytes transferred, if that is what the application requires.

Methods

FormatBytes(bytes As Integer) As String
This method allows the developer to scale, convert and format the byte count sent by Progress_FTP (or any byte value) into a string. The string will scale and append the appropriate label ("bytes", "Kb", or "Mb").

FTPMessages

All messages are stored as string constants in this developer-accessible module. It allows the FTP Suite developer to produce multiple language applications using FTP Suite and even change messages based on target platform.
Note: FTP Suite comes with German and French translations of some of these messages. The German translations have been generated by Babel Fish Translation Services by Alta Vista. The accuracy of these translations is not guaranteed by Apogee Development, Ltd. and should not be used for released software. They are for demonstration purposes only. The French translations are properly localized and can be used.

The messages are divided into the following categories:

Error Messages – Delivered by Error_FTP Event

FTP_ErrorWritingLocalFile = "Error writing local file: "
FTP_ErrorReceivingFileList = "Error while receiving the directory list: "
FTP_ErrorReceivingFile = "Error while receiving the file: "
FTP_ErrorSendingFile = "Error while sending the file: "
FTP_ErrorCannotOpenFile = "Cannot open file: "
FTP_ErrorCannotFindFile = "Cannot Find File: "
FTP_ErrorConnectionTimeout = "Error: The connection timed out. Ending session."
FTP_ErrorCannotFindPath = "Cannot find this path: "
FTP_ErrorGeneric = "Error: "
FTP_ErrorUnknown = "Unknown error during ("
FTP_ErrorNoDataSocket = "Data Socket is no longer active."
FTP_ErrorNoCommandSocket = "Command Socket is no longer active."
FTP_ErrorConnectionLost = "Connection Lost."

Completion Messages – Delivered by Successful_FTP Event

FTP_LogonComplete = "Logon Complete."
FTP_LogoffComplete = "Logoff Complete."
FTP_UserInterrupt = "FTP interrupted by user."
FTP_End = "FTP Session Complete."
FTP_FileListComplete = "The file list has been transferred."
FTP_FileRenameComplete = "File Rename Complete."
FTP_FileTransferComplete = "File Transfer Complete."
FTP_FileDeleted = "File has been deleted."
FTP_DirectoryTransferComplete = "Directory Transfer Complete."
FTP_DirectoryDeleted = "The directory has been deleted."
FTP_DirectoryRemoved = "Empty Directory Removed."
FTP_DirectoryCreated = "Directory Created."
FTP_DirectoryChangeComplete = "Directory Change Complete."
FTP_WorkingDirectoryComplete = "The working directory has been transferred."
FTP_SystemInfoComplete = "System information received."
FTP_ChmodComplete = "Chmod Command Complete."
FTP_SITECommandComplete = "SITE Command Complete."
FTP_DirectCommandComplete = "Direct Command Complete."

Internal Status Messages – Delivered by Status_FTP Event

FTP_SendUserID = "Sending UserID"
FTP_SendPassword = "Sending Password"

FTP_ChangingDirectory = "Changing directory to: "
FTP_PassiveMode = "Passive Mode"
FTP_ASCII = "Changing transfer type to ASCII."
FTP_SetDataPort = "Setting data port to: "
FTP_ChangeType = "Change transfer type."
FTP_RenameFile = "Renaming file."
FTP_ParentDirectory = "Changing to parent directory."
FTP_HostPort = "Connecting to host-supplied port."
FTP_Deleting = "Deleting: "
FTP_NextSubDir = "Next subdirectory."
FTP_GetFile = "Getting file "
FTP_NextFile = "Next file"
FTP_ActiveMode = "Active Mode"
FTP_SubDir = "Handle Subdirectory "
FTP_GetWorkingDir = "Getting working directory for "
FTP_GetDirNameList = "Getting directory name list."
FTP_GetDirList = "Getting full directory list for "
FTP_FileLoop = "Handle File Loop "
FTP_Connecting = "Connecting..."
FTP_CreateDir = "Creating directory: "
FTP_SendFile = "Sending file."
FTP_StartLoop = "Start of transfer loop."
FTP_UpDir = "Going up one directory level."
FTP_HandleParentDir = "Handle parent directory "
FTP_GetSystemInfo = "Getting system information."
FTP_GetStatus = "Getting System Status"

Send FTP Command Messages – Delivered by Status_FTP Event

FTP_SendChangeDir = "Send Change Directory Command"
FTP_SendDeleteSingleFile = "Send Delete Single File Command"
FTP_SendGetDirList = "Send Get Directory List Command"
FTP_SendGetStatus = "Send Get Status Command"
FTP_SendGetSystemInfo = "Send Get System Info Command"
FTP_SendLogoff = "Send Logoff Command"
FTP_SendLogon = "Send Logon Command"
FTP_SendGetSingleFile = "Send Get Single File Command"
FTP_SendPutSingleFile = "Send Put Single File Command"
FTP_SendRenameSingleFile = "Send Rename Single File Command"
FTP_SendRemoveDir = "Send Remove Directory Command"
FTP_SendMakeDir = "Send Make Directory Command"
FTP_SendGetWorkingDirectory = "Send Print Working Directory Command"
FTP_SendChmodSingleFile = "Send Change Mode Command"
FTP_SendDirectCommand = "Send Direct Command"
FTP_SendSITECommand = "Send SITE Command"

Start FTP Sequence Messages – Delivered by Status_FTP Event

FTP_StartDeleteDir = "Start Delete Directory Sequence"
FTP_StartDeleteFile = "Start Delete File Sequence"
FTP_StartGetDir = "Start Get Directory Sequence"
FTP_StartGetDirList = "Start Get Directory List Sequence"

FTP_StartGetFile = "Start Get File Sequence"
FTP_StartPutDir = "Start Put Directory Sequence"
FTP_StartPutFile = "Start Put File Sequence"
FTP_StartRenameFile = "Start Rename File Sequence"

Registration

Apogee Development, Ltd. offers three licenses for FTP Suite, which may be obtained at the FTP Suite site:

http://www.ftpsuite.com/register.html

Please read the license agreement included below and with the software, which outlines the specific definition of the two licenses. A license provides free upgrades for all minor version revisions.

The demo version will display a registration notice at the start of every FTP transfer session. Once registered, the notice does not appear.

Support

Apogee Development, Ltd. is committed to supporting FTP Suite for both hobbyist and professional use.

Included

Apogee Development, Ltd. actively participates in the REALbasic NUG and will often address FTP Suite issues in that forum. Join at: **http://www.realsoftware.com/support/listmanager**.

Join the FTP Suite Announcement List by at: **http://www.ftpsuite.com/list.html**. We respect your privacy and do not give your email address to *anyone*. Members of this list typically receive 6-8 emails a year announcing new versions and asking for feedback on FTP Suite features and issues.

Enter all bug reports and feature suggestions using the Support Question Form at **http://www.ftpsuite.com/support.html**. A response is guaranteed within five US business days. *Only inquires sent using this form will be answered*.

Extended Support Plans

Apogee Development, Ltd. offers the following options for **Professional Licensees *Only***:

Support Incident - $39.99/incident - 3 US business days response time guaranteed. - Apogee Development, Ltd. will review customer code, and test with customer servers, if required.

Support Plan - $149.99/year - Includes 6 Support Incidents/year - 3 US business days response time guaranteed.

Upgraded Support Plan - $299.99/year - Includes 12 Support Incidents/year – 24 hour response during business hours guaranteed

License Agreement

This is a legal agreement between you and Apogee Development, Ltd., covering your use of FTP Suite and related materials (the "Software"). Be sure to read the following agreement before using the Software.

BY USING THE SOFTWARE (REGARDLESS IF YOU HAVE PURCHASED THE SOFTWARE OR NOT), YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DO NOT USE THE SOFTWARE AND DESTROY ALL COPIES IN YOUR POSSESSION.

Reselling of the Software as a stand-alone module without the prior written consent of Apogee Development, Ltd. is strictly prohibited. Any ventures that wish to incorporate the Software as a stand-alone module into their own commercial productions/services (such as shareware CD-ROM collections, etc.) must first contact Apogee Development, Ltd. for written permission. Apogee Development, Ltd. must always be credited as the author of the Software.

By using the Software, you acknowledge that the Software and all related products constitute valuable property of Apogee Development, Ltd. and that all title and ownership rights to the Software and related materials remain exclusively with Apogee Development, Ltd. Apogee Development, Ltd. reserves all rights with respect to the Software and all related products under all applicable laws for the protection of proprietary information, including, but not limited to, trade secrets, copyright, trademarks and patents.

The Software is owned by Apogee Development, Ltd. and is protected by United States copyright laws and international treaty provisions. Therefore, you must treat the Software like any other copyrighted material (e.g. a book or musical recording). Paying the Software registration fee allows you the right to use one copy of the Software on a single computer. You may not network the Software or otherwise use it or make it available for use on more than one computer at the same time. You may not rent or lease the Software, nor may you modify, adapt, translate, reverse engineer, decompile, or disassemble the Software. If you violate any part of this agreement, your right to use this Software terminates automatically and you must then destroy all copies of the Software in your possession.

The Software and its related documentation are provided "AS IS" and without warranty of any kind and Apogee Development, Ltd. expressly disclaims all other warranties, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Under no circumstances shall Apogee Development, Ltd. be liable for any incidental, special, or consequential damages that result from the use or inability to use the Software or related documentation, even if Apogee Development, Ltd. has been advised of the possibility of such damages. In no event shall Apogee Development, Ltd's liability exceed the license fee paid, if any.

This license allows you to utilize the Software in compiled applications created with the REALbasic Integrated Development Environment. There are no royalties or additional fees beyond the original cost of the Software.

The “Hobbyist” license has specific limitations of use. It can only be used for personal projects or publicly released Freeware and Shareware compiled applications. For purposes of this license, Freeware is defined as a compiled application that has no fee associated with it, either for purchase or for support, and Shareware is a compiled application that allows the user to try the product before purchase, with no restrictions on usability.

The “Commercial” license is required for all commercial and In-House compiled applications. For purposes of this license, Commercial is defined as a compiled application that requires a fee in order to release complete functionality of the software, and In-House is defined as a compiled application that is used to support a commercial enterprise. Unauthorized distribution or resale of FTP Suite is strictly prohibited. Commercial or public use of an unregistered copy of FTP Suite is strictly prohibited.

This Agreement shall be governed by and construed in accordance with the domestic laws of the State of Ohio without giving effect to any choice or conflict of law provision or rule (whether of the State of Ohio or any other jurisdiction) that would cause the application of the laws of any jurisdiction other than the State of Ohio. Any action or proceeding arising out of or related to this Agreement shall be brought and enforced only in the state and federal courts located in Fairfield or Franklin County, Ohio, and the parties consent to the personal jurisdiction of such courts and waive any argument that venue in any such forum is not convenient.

If any party hereto brings any suit, action, counterclaim, arbitration, or other proceeding relating to the enforcement or interpretation of any of the provisions of this business transaction, or relating to the subject matter of this business transaction, the prevailing party therein shall be entitled to recover a reasonable allowance for the attorneys' fees and litigation expenses in addition to court costs. The "prevailing party" within the meaning of this agreement includes without limitation a party who: (i) agrees to dismiss an action or proceeding upon the other's payment of the sums allegedly due or performance of the obligations allegedly breached; or (ii) obtains substantially the relief that such party seeks.