Uploading and Downloading Files

The simplest way to transfer data to an FTP or HTTP server is to use the **put** command to upload, or use the URL in an expression to download.

The Internet library includes additional commands to upload and download files to and from an FTP server. These commands offer more versatile options for monitoring and controlling the progress of the file transfer.

Uploading using the put command

As mentioned above, putting something into an **ftp** or **http** URL uploads the data to the server:

```
put myVariable into URL
“ftp://user:pass@ftp.example.org/newfile.txt"
```

If you use the **put** command with a **file** or **binfile** URL as the source, the file is uploaded:

```
put URL "file:newfile.txt" into URL
“ftp://user:pass@ftp.example.org/newfile.txt"
```

When you upload data in this way, the operation is blocking: that is, the handler pauses until the upload is finished.

(See below for details on how to create a file transfer that is not blocking.)

If there is an error, the error is placed in the **result** function:

```
put field "Data" into URL myFTPDestination
if the result is not empty then beep 2
```

Important: Uploading or downloading a URL does not prevent other messages from being sent during the file transfer: the current handler is blocked, but other handlers are not.

For example, the user might click a button that uploads or downloads another URL while the first URL is still being uploaded.

In this case, the second file transfer is not performed and the **result** is set to "Error Previous request has not completed."

To avoid this problem, you can set a flag while a URL is being uploaded, and check that flag when trying to upload or download URLs to make sure that there is not already a file transfer in progress.

Downloading using a URL

Referring to an **ftp** or **http** URL in an expression downloads the document.

```
put URL "ftp://ftp.example.net/myfile.jpg" into image 1
get URL “http://www.example.com/newstuff/newfile.html"
```

If you use the **put** command with a **file** or **binfile** URL as the destination, the document is downloaded to the file:

```
put URL "ftp://ftp.example.net/myfile.jpg" into URL
    “binfile:/Disk/Folder/myfile.jpg"
```

Non-blocking transfers

When you transfer a file using URL containers, the file transfer stops the current handler until the transfer is done. This kind of operation is called a blocking operation, since it blocks the current handler as long as it's going on.

If you want to transfer data using *http* without blocking, use the **load** command.

if you want to transfer large files using *ftp*, use the **libURLftpUpload**, **libURLftpUploadFile**, or **libURLDownloadToFile** commands.

Non-blocking file transfers have several advantages:

Since contacting a server may take some time due to network lag, the pause involved in a blocking operation may be long enough to be noticeable to the user.

If a blocking operation involving a URL is going on, no other blocking operation can start until the previous one is finished. If a non-blocking file transfer is going on, however, you can start other non-blocking file transfers. This means that if you use the library commands, the user can begin multiple file transfers without errors.

During a non-blocking file transfer, you can check and display the status of the transfer. This lets you display the transfer's progress and allow the user to cancel the file transfer.

Using the load command

The **load** command downloads the specified document in the background and places it in a cache. Once a document has been cached, it can be accessed nearly instantaneously when you use its URL, because LiveCode uses the cached copy in memory instead of downloading the URL again.

To use a file that has been downloaded by the load command, refer to it using the URL keyword as usual.

When you request the original URL, LiveCode uses the cached file automatically.

For best performance, use the `load` command at a time when response speed isn't critical (such as when your application is starting up), and only use it for documents that must be displayed quickly, such as images from the web that will be shown when you go to the next card.

Checking status when using the load command

While a file is being transferred using the load commands, you can check the status of the transfer using the **URLStatus** function.

This function returns the current status of a URL that's being downloaded or uploaded:

```
local tUrl
put "ftp://ftp.example.com/myfile.txt" into tUrl
put the URLStatus of tUrl into field "Current Status”
```

The **URLStatus** function returns one of the following values:

- ***queued*** **:** on hold until a previous request to the same site is completed
- ***contacted*** **:** the site has been contacted but no data has been sent or received yet
- ***requested*** **:** the URL has been requested
- ***loading bytesTotal, bytesReceived*** **:** the URL data is being received
- ***uploading bytesTotal, bytesReceived*** **:** the file is being uploaded to the URL
- ***cached*** **:** the URL is in the cache and the download is complete
- ***uploaded*** **:** the application has finished uploading the file to the URL
- ***error*** **:** an error occurred and the URL was not transferred
- ***timeout*** **:** the application timed out when attempting to transfer the URL
-

To monitor the progress of a file transfer or display a progress bar, you check the **URLStatus** function repeatedly during the transfer.

The easiest way to do this is with timer based messaging – see the section of the same name in the *LiveCode Script* guide, for more information.

Canceling a file transfer & emptying the cache

To cancel a transfer initiated with the load command and empty the cache, use the **unload** command.

```
unload URL “http://example.org/new_beta"
```

Uploading and downloading large files using FTP

The Internet library provides a number of commands for transferring larger files via FTP without blocking.

- **libURLftpUpload** uploads data to an FTP server
- **libURLftpUploadFile** uploads a file to an FTP server
- **libURLDownloadToFile** downloads a file from an FTP server to a local file

The basic effect of these commands is the same as the effect of using URLs: that is, the data is transferred to or from the server.

However, there are several differences in how the actual file transfer is handled. Because of these differences, the library commands are more suitable for uploads and downloads, particularly if the file being transferred is large.

The following sets of statements each show one of the Internet library commands, with the equivalent use of a URL:

```
libURLftpUpload myVar,"ftp://me:pass@example.net/file.txt"
put myVar into URL "ftp://me:pass@example.net/file.txt"
```

```
libURLftpUploadFile "test.data","ftp://ftp.example.org/test"
put URL "binfile:test.data" into URL "ftp://ftp.example.org/test
```

```
libURLDownloadToFile "ftp://example.org/new_beta","/HD/File"
put URL "ftp://example.org/new_beta" into URL “binfile:/HD/File"
```

Using callback messages

When you start a file transfer using the **libURLftpUpload**, **libURLftpUploadFile**, or **libURLDownloadToFile** command, you can optionally specify a callback message, which is usually a custom message that you write a handler for.

This message is sent whenever the file transfer's **URLStatus** changes, so you can handle the callback message to handle errors or to display the file transfer's status to the user.

The following simple example demonstrates how to display a status message to the user.

The following handlers might be found in a button's script:

```
on mouseUp
    local tUrl
    put "ftp://example.org/new_beta" into tUrl
    libURLDownloadToFile tUrl,"/HD/Latest Beta","showStatus"
end mouseUp

on showStatus theURL
    put the URLStatus of theURL into field "Status"
end showStatus
```

When you click the button, the **mouseUp** handler is executed.

The **libURLDownloadToFile** command begins the file transfer, and its last parameter specifies that a *showStatus* message will be sent to the button whenever the **URLStatus** changes.

As the **URLStatus** changes periodically throughout the download process, the button's *showStatus* handler is executed repeatedly.

Each time a *showStatus* message is sent, the handler places the new status in a field. The user can check this field at any time during the file transfer to see whether the download has started, how much of the file has been transferred, and whether there has been an error.

If a file transfer was started using the **libURLftpUpload**, **libURLftpUploadFile**, or **libURLDownloadToFile** command, you can cancel the transfer using the **unload** command.

Uploading, downloading, and memory

When you use a URL as a container, LiveCode places the entire URL in memory. For example, if you download a file from an FTP server using the **put** command, LiveCode downloads the whole contents of the file into memory before putting it into the destination container.

If the file is too large to fit into available memory, a file transfer using this method will fail (and may cause other unexpected results).

The library commands **libURLftpUpload**, **libURLftpUploadFile**, and **libURLDownloadToFile**, however, do not require the entire file to be loaded into memory.

Instead, they transfer the file one piece at a time. If a file is (or might be) too large to comfortably fit into available memory, you should always use the library commands to transfer it.

Using a stack on a server

Ordinarily, you use stack files that are located on a local disk. You can also open and use a stack that is located on an FTP or HTTP server.

Using this capability, you can update an application by downloading new stacks, make new functionality available via the Internet, and even keep most of your application on a server instead of storing it locally.

Going to a stack on a server:

As with local stack files, you use the **go** command to open a stack that's stored on a server:

```
go stack URL "http://www.example.org/myapp/main.rev"
go stack URL “ftp://user:pass@example.net/secret.rev"
```

Note: For such a statement to work, the stack file must have been uploaded as binary data, uncompressed, and not use encodings such as BinHex.

Tip: If you need to download a large stack, use the **load** command to complete the download before using the **go** command to display the stack.

This allows you to display a progress bar during the download.

LiveCode automatically downloads the stack file. The main stack of the stack file then opens in a window, just as though you had used the **go** command to open a local stack file.

You can go directly to a specific card in the stack:

```
local tStackUrl
put "http://www.example.org/myapp/main.rev" into tStackUrl
go card "My Card" of stack URL tStackUrl
```

To open a substack instead, use the substack's name:

```
local tStackUrl
put "http://www.example.org/myapp/main.rev" into tStackUrl
go stack "My Substack" of stack URL tStackUrl
```

Using a compressed stack

You cannot directly open a stack that's compressed. However, since the stack URL is a container, you can use the URL as the parameter for the **decompress** function.

The function takes the stack file data and decompresses it, producing the data of the original stack file. You can open the output of the function directly as a stack.

The following statement opens a compressed stack file on a server:

```
go decompress(stack URL “http://www.example.net/comp.gz")
```

The statement automatically downloads the file "comp.gz", uncompresses it, and opens the main stack of the file.

Saving stacks from a server

When a stack is downloaded using the **go** command, it's loaded into memory, but not saved on a local disk. Such a stack behaves like a new (unsaved) stack until you use the **save** command to save it as a stack file.

Note: Saving a stack that has been downloaded with the **go** command does not re-upload it to its server. To upload a changed stack, you must save it to a local file, then use one of the methods described in this topic to upload the file to the server.