

Examples

```
// Unzip a file using full paths
String zipFileString = "C:Users:Fred:Documents:Test.zip"
String outputPathString = "C:Users:Fred:Documents:Extracted"
UnzipFile zipFileString, outputPathString

// Unzip a file using symbolic paths
String zipFileName = "Test.zip"
String outputFolderName = "" // Unzip to folder specified by /POUT
UnzipFile/PIN=home/POUT=home zipFileName, outputFolderName
```

UpperStr**UpperStr (str)**

The UpperStr function returns a string expression in which all lower-case ASCII characters in *str* are converted to upper-case.

See Also

The **LowerStr** function.

URLDecode**URLDecode (inputStr)**

The URLDecode function returns a percent-decoded copy of the percent-encoded string *inputStr*. It is unlikely that you will need to use this function; it is provided for completeness.

For an explanation of percent-encoding, see **Percent Encoding** on page IV-268.

Example

```
String theURL = "http://google.com?key1=35%25%20larger"
theURL = URLDecode(theURL)
Print theURL
http://google.com?key1=35% larger
```

See Also

URLEncode, **URLRequest**, **URLs** on page IV-267.

URLEncode**URLEncode (inputStr)**

The URLEncode function returns a percent-encoded copy of *inputStr*.

Percent-encoding is useful when encoding the query part of a URL or when the URL contains special characters that might otherwise be misinterpreted by a web server. For an explanation of percent-encoding, see **Percent Encoding** on page IV-268.

Example

```
String baseURL = "http://google.com"
String key1 = "key1"
String value1 = URLEncode("35% larger")
String theURL = ""
sprintf theURL, "%s%s=%s", baseURL, key1, value1
Print theURL
http://google.com?key1=35%25%20larger
```

See Also

URLDecode, **URLRequest**, **URLs** on page IV-267.

URLRequest

URLRequest [flags] url=urlStr [method=methodName, headers=headersStr]

The URLRequest operation connects to a URL using the specified method and optionally stores the response from the server. *urlStr* can point to a remote server or to a local file. The server's response is stored in the *S_serverResponse* output variable or in a file if you use the */FILE* flag.

The URLRequest operation was added in Igor Pro 7.00.

Keywords

The `url=urlStr` keyword is require. All others are optional.

- `url=urlStr`

A string containing the URL to retrieve. See **URLs** on page IV-267 for details.
- `method=methodName`

Specifies which method to use for the request. The get method is used by default. This table shows the valid methods for each supported scheme. Not all servers support all of the listed methods.

Scheme	Supported Methods
http, https	get, post, put, head, delete
ftp	get, put
file	get, put

- `headers=headersStr`

Because `methodName` is a name, not a string, you must not enclose it in quotes. Before using the post method, you should read **The HTTP POST Method** on page V-1058 so that you know how to use the optional headers parameter. If you use the head method, URLRequest sets the `S_serverResponse` output variable to "" because only the headers are retrieved. As with other methods, the headers are stored in the `S_headers` output variable.

Specifies a string containing additional or replacement headers to use with the request. This parameter is ignored unless the scheme is http or https. The headers parameter is provided primarily for use with the post method when making HTTP requests and is ignored for schemes other than http/https. The header consists of a colon-separated key:value pair (though see the next paragraph for an exception). Pairs must be separated by a carriage return (`\r`) character. Certain standard headers (such as Content-Type and User-Agent) may automatically be set when making the request. You can override those default values by using this keyword and setting a different value. If you add a header with no content, as in "Accept:" (there is no data on the right side of the colon), the internally used header is disabled. To actually add a header with no content, use the form "MyHeader;" (note the trailing semicolon). Any headers specified with this keyword are sent only to the http server, not to the proxy server, if one is in use. See **The HTTP POST Method** on page V-1058 for a detailed explanation and examples.

Flags

- `/AUTH={username, password }`

Uses the specified `username` and `password` string parameters for authentication. Values provided here override any username and/or password provided as part of the URL. To specify a username but not a password, pass "" for the `password` parameter.

Note: See **Safe Handling of Passwords** on page IV-270 for more information on how to use URLRequest to prevent authentication information, such as passwords, from being accidentally revealed.

<code>/DFIL=<i>dataFileNameStr</i></code>	<p>Specifies a file name to use as a source of data. Typically this flag is used only with the put or post methods, but it is accepted with all methods. Unless <i>dataFileNameStr</i> is a full path, the <code>/P</code> flag must also be used. When using the post or put methods, one and only one of the <code>/DFIL</code> or <code>/DSTR</code> flags must be used.</p> <p>When you use <code>/DFIL</code> and the method is anything other than put, the "Content-Type: application/x-www-form-urlencoded" header is automatically added. If necessary, you can override this behavior using the headers keyword. See The HTTP POST Method on page V-1058 for more information.</p>
<code>/DSTR=<i>dataStr</i></code>	<p>Specifies the string to use as a source of data. Typically you use this flag only with the put or post methods, but it is accepted with all methods. When using the post or put methods, one and only one of the <code>/DFIL</code> or <code>/DSTR</code> flags must be used.</p> <p>When <code>/DSTR</code> is used and the method is anything other than put, the "Content-Type: application/x-www-form-urlencoded" header is automatically added. If necessary, you can override this behavior using the headers keyword. See The HTTP POST Method on page V-1058 for more information.</p>
<code>/FILE=<i>destFileNameStr</i></code>	<p>If present, URLRequest saves the server's response in a file instead of in the <code>S_serverResponse</code> output variable.</p> <p>URLRequest ignores <code>/FILE</code> if you include the <code>/IGN</code> flag and the <i>ignoreResponse</i> parameter is not 0.</p> <p><i>destFileNameStr</i> can be a full path to the file, in which case <code>/P=<i>pathName</i></code> is not needed, a partial path relative to the folder associated with <i>pathName</i>, or the name of a file in the folder associated with <i>pathName</i>. If the file already exists, URLRequest returns an error unless you include the <code>/O</code> flag.</p> <p>If you include <code>/O</code> and the file already exists, the existing file is overwritten. This happens even in the event of an empty response or transfer error.</p> <p>You should consider using the <code>/FILE</code> flag when you are expecting the server to return a large amount of data, such as when downloading a file.</p>
<code>/IGN[=<i>ignoreResponse</i>]</code>	<p>Ignore and do not store the server's response to the request. <code>/IGN</code> alone has the same effect as <code>/IGN=1</code>.</p> <p>If <i>ignore</i> is turned on, URLRequest sets the <code>S_serverResponse</code> output variable to "", regardless of whether the server responded to the request or not. If you include the <code>/FILE</code> flag, URLRequest does not create the output file and sets <code>S_fileName</code> is set to "".</p> <p>This flag is useful only when your goal is to establish a connection with a server and the server's response is not important. All error message codes and strings are still set when <i>ignore</i> is on.</p> <p><code>/IGN=0</code>: Same as no <code>/IGN</code>.</p> <p><code>/IGN=1</code>: Ignore server response completely. <code>S_headers</code> is set to "".</p> <p><code>/IGN=2</code>: Ignore server response but capture the headers of the response. <code>S_serverResponse</code> is set to "" but <code>S_headers</code> will contain the headers.</p>
<code>/NRED=<i>maxNumRedirects</i></code>	<p>Specifies the maximum number of redirects that are allowed. A redirect means that when a certain URL is requested, the server responds telling the client to try a different URL. Most web browsers automatically follow server redirects, up to a certain limit. For security purposes, it is sometimes useful to prevent redirects from being followed at all.</p> <p><i>maxNumRedirects</i> is a number between -1 and 1000. To allow infinite redirection, set <i>maxNumRedirects</i> to -1. If you omit the <code>/NRED</code> flag, a moderate value (currently 20, but subject to change in the future) is used.</p>

<code>/O</code>	Specifies that the file is to be overwritten when you use the <code>/FILE</code> flag and the output file already exists. If you omit <code>/O</code> and the file exists, URLRequest returns an error.
<code>/P=pathName</code>	<p>Specifies the folder to use for the output file, specified by the <code>/FILE</code> flag, and/or the source data file, specified by the <code>/DFIL</code> flag. <code>pathName</code> is the name of an existing Igor symbolic path.</p> <p>The <code>/P</code> flag affects both the <code>/FILE</code> and <code>/DFIL</code> flags. If you use both flags and want to use different directories, you must provide a full path for one or both of the <code>/FILE</code> and <code>/DFIL</code> flag parameters.</p> <p>If the <code>/P</code> flag is used without one or both of the <code>/FILE</code> and <code>/DFIL</code> flags, it is ignored.</p>
<code>/PROX[={proxyURLStr, proxyUserNameStr, proxyPassStr, proxyOptions}]</code>	<p>NOTE: This flag is experimental and has not been extensively tested. The behavior of this flag may change in the future, or it may be eliminated entirely.</p> <p>Designates a proxy server to be used when making the connection. <i>proxyURLStr</i> is either the host name or IP address of the proxy server, or a full URL. If a full URL is used, the scheme specifies which kind of proxy is used. Typically the scheme for a proxy server should be either <code>http</code> or <code>socks5</code>. If no scheme is provided, <code>http</code> is assumed. See URLs on page IV-267 for more information.</p> <p>If the proxy server does not require authentication, or if the username and password for the proxy server are specified in <i>proxyURLStr</i>, the optional <i>proxyUserNameStr</i> and <i>proxyPassStr</i> parameters do not need to be provided. The following two examples do the same thing:</p> <pre>/PROX={"http://proxy.example.com:800"} /PROX={"http://proxy.example.com:800", "", ""}</pre> <p>As with other URLs, you can also provide the username and password as part of the URL itself. The following two examples do the same thing:</p> <pre>/PROX={"http://proxy.example.com:800", "user", "pass"} /PROX={"http://user:pass@proxy.example.com:800"}</pre> <p><i>proxyOptions</i> is optional and for future use. It is currently ignored. If provided, you must set it to 0.</p> <p>If you use the <code>/PROX</code> flag without any parameters, Igor attempts to get proxy information from the operating system's proxy configuration information. If Igor cannot get any proxy server information from the system, no proxy server is used.</p> <p>See Safe Handling of Passwords on page IV-270 for more information on how to use URLRequest to prevent authentication information, such as passwords, from being accidentally revealed.</p>
<code>/TIME=timeoutSeconds</code>	<p>Forces the operation to time out after <i>timeoutSeconds</i> seconds if it has not completed by that time. If this flag is not provided, URLRequest runs until the server has finished sending and receiving data. For simple requests a value of a few seconds is appropriate. However, because uploading and/or downloading large amounts of data may take a long time, if <i>timeoutSeconds</i> is too small the request might be prematurely terminated. If you omit the <code>/TIME</code> flag, or if <i>timeoutSeconds</i> = 0, there is no timeout.</p> <p>Regardless of whether or not you include <code>/TIME</code>, you can abort the operation by pressing the User Abort Key Combinations.</p>

<i>/V=diagnosticMode</i>	<p>This flag is useful only when your goal is to establish a connection with a server and the server's response is not important. All error message codes Controls diagnostic messages printed in the history area of the command window.</p> <p><i>/V=0:</i> Do not print any diagnostic messages. This is the default if you omit <i>/V</i>.</p> <p><i>/V=1:</i> Prints an error message if a run time error occurs.</p> <p><i>/V=2:</i> Prints full debugging information.</p>
<i>/Z[=z]</i>	<p>Suppress error generation. Use this if you want to handle errors yourself.</p> <p><i>/Z=0:</i> Do not suppress errors. This is the default if <i>/Z</i> is omitted.</p> <p><i>/Z=1:</i> Any errors generated by Igor do not stop execution. If there is an error the error code is stored in <i>V_Flag</i>.</p> <p><i>/Z</i> alone has the same effect as <i>/Z=1</i>.</p>

Output Variables

URLRequest sets the following output variables:

<i>V_flag</i>	<p><i>V_flag</i> is zero if the operation succeeds without an error or a non-zero Igor error code if it fails.</p> <p>Note that "succeeds without an error" does not necessarily mean that the operation performed as you intended. For example, if you attempt to connect to a server that requires a username and password for authentication but you do not provide this information, <i>V_flag</i> is likely to be 0, indicating no error. You need to inspect the value of <i>V_responseCode</i> to ensure that it is what you expect.</p>
<i>V_responseCode</i>	<p>Contains the status code provided by the server.</p> <p><i>V_responseCode</i> is valid only if <i>V_flag</i> is 0, meaning that no error occurred. If <i>V_flag</i> is nonzero, an error occurred and <i>V_responseCode</i> will be 0.</p> <p>Different schemes use different sets of status codes.</p> <p>A list of http/https status codes and their definitions can be found at: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html</p> <p>A list of FTP status codes can be found at: https://en.wikipedia.org/wiki/List_of_FTP_server_return_codes</p>
<i>S_serverResponse</i>	<p>Contains the server's response to the request.</p> <p><i>S_serverResponse</i> is set to "" if you use the <i>/FILE</i> flag or the <i>/IGN=1</i> or <i>/IGN=2</i> flags.</p> <p>We recommend that you use the <i>/FILE</i> flag when you expect the size of the server response to be large, such as when downloading a large file.</p>
<i>S_headers</i>	<p>Contains a list of all of the headers received as part of the response. Header names are separated from their value by a colon and pairs are separated by a carriage return followed by a line feed (<code>\r\n</code>). For schemes other than http and https, <i>S_headers</i> is set to "".</p>
<i>S_fileName</i>	<p>Contains the full Igor-style path to the output file if the <i>/FILE</i> flag was used. If there was an error saving to the specified file, or another kind of error, or if the <i>/FILE</i> flag was not used, <i>S_fileName</i> is set to "".</p>

Basic Examples

```
// Retrieve the contents of the WaveMetrics home page.
// Output is stored in S_serverResponse
URLRequest url="http://www.wavemetrics.com"

// Download the Windows Igor6 installer and save it as a file on the desktop.
```

```
NewPath/O Desktop, SpecialDirPath("Desktop", 0, 0, 0)
String theURL = "http://www.wavemetrics.net/Downloads/Win/setupIgor6.exe"
URLRequest/FILE="setupIgor6.exe"/P=Desktop url=theURL

// Download a file to the desktop from an FTP server.
NewPath/O Desktop, SpecialDirPath("Desktop", 0, 0, 0)
String theURL = "ftp://ftp.wavemetrics.com/test/test.htm"
URLRequest/O/FILE="test.htm"/P=Desktop url=theURL

// Upload the same file to the FTP server.
theURL = "ftp://user:pass@ftp.wavemetrics.com/test.htm"
URLRequest/DFIL="test.htm"/P=Desktop method=put, url=theURL
```

Using the File Scheme

URLRequest supports the file scheme in URLs. You must provide the full path to the file as a native Windows-style or UNIX-style path, depending on which platform the code is running.

You must specify the schema as "file:///". Note that you must use three front slash characters (/) between the colon and the full file path (for most URLs you would only use two slashes).

```
URLRequest url="file:///C:\\Data\\Trial1\\control.ibw" // Works on Windows only
URLRequest url="file:///Users/bob/Data/Trial1/control.ibw" // Works on Macintosh only
URLRequest url="file:///C:\\Data\\Trial1\\control.ibw" // Doesn't work
URLRequest url="file:///Users/bob/Data/Trial1/control.ibw" // Doesn't work
```

The following example shows how to convert a full Igor file path to a native path suitable for use as a file URL:

```
String nativeFilePath
#ifdef WINDOWS
    String igorFilePath = "C:Documents:myFile.txt"
    nativeFilePath = ParseFilePath(5, igorFilePath, "*", 0, 0)
#endif
#ifdef MACINTOSH
    String igorFilePath = "MacintoshHD:Documents:myFile.txt"
    nativeFilePath = ParseFilePath(5, igorFilePath, "/", 0, 0)
    // Remove the leading "/" from nativeFilePath. Otherwise,
    // when we prepend "file:/" below, we'd end up with four
    // slashes, which isn't allowed.
    nativeFilePath = nativeFilePath[1, INF]
#endif
URLRequest url="file:/" + nativeFilePath
```

The HTTP POST Method

Like the HTTP GET method, the HTTP POST method can be used to transmit information to a web server. When using the GET method, all information must be contained within the URL itself. As an example, making a GET request to the URL <http://www.google.com/search?q=Igor+Pro+WaveMetrics> searches Google using the keywords "Igor", "Pro", and "WaveMetrics".

Unlike the GET method, the POST method allows the client to send information to the server that is not contained within the URL itself. This is necessary in many situations, such as when uploading a file or transferring a large amount of data. In addition, because the data contained in POST requests is typically not stored in the log files of web servers, it is more appropriate for sending data that is secure, such as login credentials and form submissions which might contain sensitive information.

When using the POST method, the client must encode its data using one of several methods and must tell the server what type of data is being sent and how it is encoded. The client does this by setting the Content-Type header that is part of the request. The most commonly used content type is "application/x-www-form-urlencoded". Unless you provide your own Content-Type header using the optional headers parameter, URLRequest will set this header for you. This is true regardless of which data source (/DSTR for string or /DFIL for file) you use for the POST.

Here is a simple example that uses the POST method with URL encoded data (the default Content-Type).

```
String theURL = "http://www.example.com/process.php"
String nameData = URLEncode("name") + "=" + URLEncode("Dan P. Ikes, Jr.")
String address = "650 E. Chicago Ave."
String addressData = URLEncode("address") + "=" + URLEncode(address)
String postData = nameData + "&" + addressData
URLRequest/DSTR=postData method=post, url=theURL
```

There are two things to note in this example.

First, both the key name and value string are passed through **URLEncode** so that any special characters can be percent-encoded.

Second, keys and values are separated by an equal sign ("=") and key/value pairs are separated by an ampersand ("&"). These characters are not passed through URLEncode because doing so would cause them to lose their meaning as special characters.

For more information on using the application/x-www-form-urlencoded content type, see <http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4.1>

It is also possible to transmit more complicated data using the POST method, such as simulating a form that contains regular text fields as well as a file upload field. To do so you must set the Content-Type header using the headers parameter. You must also build your post data using a multi-part header. Here is an example:

```
String theURL = "http://www.example.com/process.php"
String postData = ""
// Note: The boundary is arbitrary. It needs to be
// unique enough that it is not contained within any
// of the actual data being transmitted in the post.
String boundary = "AaBbCcDd0987"

// name
postData += "--" + boundary + "\r\n" // beginning of this part
postData += "Content-Disposition: form-data; name=\"name\"\r\n"
postData += "\r\n"
postData += "Dan P. Ikes, Jr.\r\n"

// address
postData += "--" + boundary + "\r\n"
postData += "Content-Disposition: form-data; name=\"address\"\r\n"
postData += "\r\n"
postData += "650 E. Chicago Ave.\r\n"

// file
// Open the file we plan to send and store the binary
// contents of the file in a string.
Variable refNum
Open/R/Z/P=Igor refNum as "ReadMe.ihf"
FStatus refNum
Variable fileSize = V_logEOF
String fileContents = ""
fileContents = PadString(fileContents, fileSize, 0)
FBinRead refNum, fileContents
Close refNum
postData += "--" + boundary + "\r\n"
postData += "Content-Disposition: form-data; name=\"file\"; filename=\"ReadMe.ihf\"\r\n"
postData += "Content-Type: application/octet-stream\r\n"
postData += "Content-Transfer-Encoding: binary\r\n"
postData += "\r\n"
postData += fileContents + "\r\n"

postData += "--" + boundary + "--\r\n" // end of this part

String headers
headers = "Content-Type: multipart/form-data; boundary=" + boundary
URLRequest/DSTR=postData method=post, url=theURL, headers=headers
```

For more information on using the multipart/form-data content type, see <http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4.2>

Handling JSON Data

JSON (JavaScript Object Notation) is a lightweight data-interchange format that is commonly used when interacting with websites - see <https://www.json.org/json-en.html>. For full read and write JSON support in Igor Pro, we recommend using the JSON XOP, developed by byte physics. You can download it from <https://docs.byte-physics.de/json-xop>.

See Also

URLEncode, **URLDecode**, **Base64Encode**, **Base64Decode**, **FetchURL**, **BrowseURL**

Network Communication on page IV-267, **URLs** on page IV-267