This chapter describes the functions, structures, and data types that you can use to use the URL Access Manager in your application.

URL Access Manager Functions

The URL Access Manager functions are described in these sections:

- "Getting Information About the URL Access Manager" (page 17)
- "Downloading From and Uploading to a URL Synchronously" (page 18)
- "Controlling an Asynchronous URL Upload or Download" (page 27)
- "Getting and Setting URL Properties" (page 35)
- "URL Access Manager Utility Functions" (page 38)
- "URL Access Manager Application-Defined Routines" (page 43)

Getting Information About the URL Access Manager

Before attempting to call the URL Access Manager functions, you must make sure that the URL Access Manager is installed and that its version is compatible with your application.

URLGetURLAvailable

Determines whether the URL Access Manager is available.

```
Boolean URLGetURLAvailable ();
```

function result The `URLGetURLAvailable` function returns `TRUE` if the URL Access Manager is available; otherwise, it returns `FALSE`.

URLGetURLAccessVersion

Determines the version of the URL Access Manager.

```
OSStatus URLGetURLAccessVersion (UInt32* returnVers);
```

`returnVers` A pointer to an unsigned 32-bit integer value. On return, this value contains the version number of the URL Access Manager.

function result A result code. For a list of possible result codes, see "Result Codes" (page 59).

Downloading From and Uploading to a URL Synchronously

You can use the following synchronous functions to download and upload information from a URL:

- `URLSimpleDownload` (page 18) downloads data from a URL into a file or directory.
- `URLSimpleUpload` (page 21) uploads data from a file or directory to a URL.
- `URLDownload` (page 23) uploads data from a file or directory to a URL.
- `URLUpload` (page 25) uploads data from a file or directory to a URL.

URLSimpleDownload

Downloads data synchronously from a URL.

```
OSStatus URLSimpleDownload (
                    const char* url,
                    FSSpec* destination,
                    Handle destinationHandle,
                    URLOpenFlags openFlags,
                    URLSystemEventProcPtr eventProc,
                    void* userContext);
```

`url` A pointer to the URL from which data is to be downloaded.

`destination` A pointer to a structure of type `FSSpec` that describes the file or directory into which data is to be downloaded, or `NULL`, in which case you must supply a value for `destinationHandle` that is a valid `Handle`.

If `destination` is an `FSSpec`, but it does not specify the name of a file or directory, the name of the file or directory specified by the URL is used. If that file or directory already exists and you do not specify `kURLReplaceExistingFlag` in the `openFlags` parameter, `URLSimpleDownload` creates a new file or directory whose name has a number appended before the extension. For example, if `destination` specifies a file named `file.txt`, `URLSimpleDownload` changes the filename to `file1.txt`.

For more information about the `FSSpec` structure, see *Inside Macintosh: Files*.

`destinationHandle`
A handle for downloading data into memory, or `NULL`, in which case you must specify a value for destination that is a valid `FSSpec`. Before calling `URLSimpleDownload`, create a handle of zero size.

`openFlags` A value of type `URLOpenFlags` that specifies download options, such as expanding a file or replacing a file if its filename is already in use. The following constants can be used to specify download options:

```
kURLReplaceExistingFlag
kURLExpandFile Flag
kURLDisplayProgressFlag
kURLDisplayAuthFlag
kURLIsDirectoryHintFlag
kURLDoNotTryAnonymousFlag
kURLDirectoryListingFlag
```

See “URL Open Flag Constants” (page 47) for descriptions of these constants.

`eventProc` A value of type `URLSystemEventProcPtr` which points to an application-defined system event callback routine (page 3-45) that the URL Access Manager calls to communicate system events to your application during the download process. If your

application requests the display of a progress indicator or authentication text fields, the progress indicator or authentication text fields appear in a movable modal dialog box.

The value of `eventProc` can be null, in which case your application will not be informed of system events. If your application requests the display of a progress indicator or authentication text fields, the progress indicator or authentication text fields appear in a non-movable modal dialog box.

`userContext` An untyped pointer to arbitrary data that the URL Access Manager passes to the application-defined system event callback routine specified by `eventProc`. Your application can use `userContext` to associate a callback with a particular call to `URLSimpleDownload`.

function result A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The `URLSimpleDownload` function downloads data synchronously from a specified URL to a specified file or directory and does not return until the download is complete. The `URLSimpleDownload` function yields time to other threads. Your application should call `URLSimpleDownload` from a thread other than the main thread so that other processes have time to run.

If your application has multiple threads, it can call `URLSimpleDownload` multiple times, but if it calls `URLSimpleDownload` with `kURLDisplayProgressFlag` set in `openFlags` when the URL Access Manager is already displaying a modal dialog box, `URLSimpleDownload` returns the error `kURLProgessAlreadyDisplayedError`.

If you want a progress indicator to be displayed during the download, specify `kURLDisplayProgressFlag` in the `openFlags` parameter. The URL Access Manager uses a modal dialog box to display the progress indicator.

Call `URLDownload` (page 23) if you need to set properties or `URLOpen` (page 27) if you need to control the download process.

URLSimpleUpload

Uploads a file or directory synchronously to an FTP URL.

```
OSStatus URLSimpleUpload (
                    char* url,
                    FSSpec* source,
                    URLOpenFlags openFlags,
                    URLSystemEventProcPtr eventProc,
                    void* userContext);
```

`url` A pointer to the FTP URL to which a file or directory is to be uploaded. To specify the name of the resulting file or directory, set `url` to a fully specified URL that does not end with a forward slash (/) character. If the file or directory exists and you want to replace it, specify the path to the destination directory, terminate the path with a forward slash (/), and specify `kURLReplaceExistingFlag` in the `openFlags` parameter. If you specify a name that already exists on the server and you do not specify `kURLReplaceExistingFlag` in the `openFlags` parameter, `URLSimpleUpload` returns the error `kURLDestinationExistsError`. If you do not specify a name, do not specify `kURLReplaceExistingFlag`, and the name already exists on the server, the URL Access Manager creates a unique name by appending a number to the original name before the extension, if any.

`source` A pointer to a structure of type `FSSpec`. Before calling `URLSimpleUpload`, set the structure to specify the file or directory you want to upload. See *Inside Macintosh: Files* for more information about the `FSSpec` structure.

`openFlags` A value of type `URLOpenFlags` that specifies upload options. The following constants can be used to specify upload options:

```
kURLReplaceExistingFlag
kURLBinHexFileFlag
kURLDisplayProgressFlag
kURLDisplayAuthFlag
kURLDoNotTryAnonymousFlag
```

See "URL Open Flag Constants" (page 47) for descriptions of these constants.

`eventProc`	A value of type `URLSystemEventProcPtr` which points to an application-defined system event callback routine (page 3-45) that the URL Access Manager calls to communicate system events to your application during the upload process. If your application requests the display of a progress indicator or authentication text fields, the progress indicator or authentication text fields appear in a movable modal dialog box. The value of `eventProc` can be null, in which case your application will not be informed of system events. If your application requests the display of a progress indicator or authentication text fields, the progress indicator or authentication text fields appear in a non-movable modal dialog box.
`userContext`	An untyped pointer to arbitrary data that the URL Access Manager passes to the application-defined system event callback routine specified by `eventProc`. Your application can use `userContext` to associate a callback with a particular call to `URLSimpleUpload`.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The `URLSimpleUpload` function uploads data synchronously to the specified FTP URL from the specified file or directory and does not return until the upload is complete. The `URLSimpleUpload` function yields time to other threads. Your application should call `URLSimpleUpload` from a thread other than the main thread so that other processes have time to run.

If your application has multiple threads, it can call `URLSimpleUpload` multiple times, but if it calls `URLSimpleUpload` with `kURLDisplayProgressFlag` set in `openFlags` when the URL Access Manager is already displaying a modal dialog box, `URLSimpleUpload` returns the error `kURLProgessAlreadyDisplayedError`.

If you want a progress indicator to be displayed during the upload, specify `kURLDisplayProgressFlag` in the `openFlags` parameter. The URL Access Manager uses a modal dialog box to display the progress indicator.

Call `URLUpload` (page 25) if you need to set properties or `URLOpen` (page 27) if you need to control the upload process.

URLDownload

Downloads data synchronously from a URL into a file or directory using a URL reference.

```
OSStatus URLDownload (
                    URLReference urlRef,
                    FSSpec* destination,
                    Handle destinationHandle,
                    URLOpenFlags openFlags,
                    URLSystemEventProcPtr eventProc,
                    void* userContext);
```

`urlRef` A reference of type `URLReference` (page 46) that specifies to URL to be downloaded. Call `URLNewReference` (page 39) to create `urlRef`.

`destination` A pointer to a structure of type `FSSpec` that describes the file or directory into which data is to be downloaded, or `NULL`, in which case you must supply a value for `destinationHandle` that is a valid `Handle`.

If `destination` is an `FSSpec`, but it does not specify the name of a file or directory, the name of the file or directory specified by the URL is used. If that file or directory already exists and you do not specify `kURLReplaceExistingFlag` in the `openFlags` parameter, `URLDownload` creates a new file or directory whose name has a number appended before the extension. For example, if `destination` specifies a file named `file.txt`, `URLDownload` changes the filename to `file1.txt`.

For more information about the `FSSpec` structure, see *Inside Macintosh: Files*.

`destinationHandle`
A handle for downloading data into memory, or `NULL`, in which case you must specify a value for destination that is a valid `FSSpec`. Before calling `URLDownload`, create a handle of zero size.

`openFlags` A value of type `URLOpenFlags` that specifies download options, such as expanding a file or replacing a file if its filename is already in use. The following constants can be used to specify download options:

kURLReplaceExistingFlag
kURLExpandFileFlag
kURLDisplayProgressFlag
kURLDisplayAuthFlag
kURLIsDirectoryHintFlag
kURLDoNotTryAnonymousFlag
kURLDirectoryListingFlag

See "URL Open Flag Constants" (page 47) for descriptions of these constants.

`eventProc` A value of type `URLSystemEventProcPtr` which points to an application-defined system event callback routine (page 3-45) that the URL Access Manager calls to communicate system events to your application during the download process. If your application requests the display of a progress indicator or authentication text fields, the progress indicator or authentication text fields appear in a movable modal dialog box.

The value of `eventProc` can be null, in which case your application will not be informed of system events. If your application requests the display of a progress indicator or authentication text fields, the progress indicator or authentication text fields appear in a non-movable modal dialog box.

`userContext` An untyped pointer to arbitrary data that the URL Access Manager passes to the application-defined system event callback routine specified by `eventProc`. Your application can use `userContext` to associate a callback with a particular call to `URLDownload`.

function result A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The `URLDownload` function downloads data synchronously from a specified URL to a specified file or directory and does not return until the download is complete. The `URLDownload` function yields time to other threads. Your application should call `URLDownload` from a thread other than the main thread so that other processes have time to run.

If you want a progress indicator to be displayed during the download, specify `kURLDisplayProgressFlag` in the `openFlags` parameter. The URL Access Manager uses a modal dialog box to display the progress indicator. If your application has multiple threads, it can call `URLDownload` multiple times, but if it calls `URLDownload` with `kURLDisplayProgressFlag` set in `openFlags` when the URL Access Manager is already displaying a modal dialog box, `URLDownload` returns the error `kURLProgessAlreadyDisplayedError`.

Call `URLOpen` (page 27) if you need to control the download process.

URLUpload

Uploads a file or directory synchronously to an FTP URL using a URL reference.

```
OSStatus URLSimpleUpload (
                    URLReference urlRef,
                    const FSSpec* source,
                    URLOpenFlags openFlags,
                    URLSystemEventProcPtr eventProc,
                    void* userContext);
```

`urlref` A reference of type `URLReference` (page 46) that specifies the URL to be uploaded. Call `URLNewReference` (page 39) to create `urlRef`.

`source` A pointer to a structure of type `FSSpec`. Before calling `URLUpload`, set the structure to specify the file you want to upload. See *Inside Macintosh: Files* for more information about the `FSSpec` structure.

`openFlags` A value of type `URLOpenFlags` that specifies upload options. The following constants can be used to specify upload options:

```
kURLReplaceExistingFlag
kURLBinHexFileFlag
kURLDisplayProgressFlag
kURLDisplayAuthFlag
kURLDoNotTryAnonymousFlag
```

See "URL Open Flag Constants" (page 47) for descriptions of these constants.

`eventProc`	A value of type `URLSystemEventProcPtr` which points to an application-defined system event callback routine (page 3-45) that the URL Access Manager calls to communicate system events to your application during the upload process. If your application requests the display of a progress indicator or authentication text fields, the progress indicator or authentication text fields appear in a movable modal dialog box. The value of `eventProc` can be null, in which case your application will not be informed of system events. If your application requests the display of a progress indicator or authentication text fields, the progress indicator or authentication text fields appear in a non-movable modal dialog box.
`userContext`	An untyped pointer to arbitrary data that the URL Access Manager passes to the application-defined system event callback routine specified by `eventProc`. Your application can use `userContext` to associate a callback with a particular call to `URLUpload`.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The `URLUpload` function uploads data synchronously to the specified FTP URL from the specified file or directory and does not return until the upload is complete. The `URLUpload` function yields time to other threads. Your application should call `URLUpload` from a thread other than the main thread so that other processes have time to run.

If you want a progress indicator to be displayed during the upload, specify `kURLDisplayProgressFlag` in the `openFlags` parameter. The URL Access Manager uses a modal dialog box to display the progress indicator. If your application has multiple threads, it can call `URLUpload` multiple times, but if it calls `URLUpload` with `kURLDisplayProgressFlag` set in `openFlags` when the URL Access Manager is already displaying a modal dialog box, `URLUpload` returns the error `kURLProgessAlreadyDisplayedError`.

Call `URLOpen` (page 27) if you need to control the upload process.

Controlling an Asynchronous URL Upload or Download

You can use the functions in this section to control the progress of a file upload or download. The functions described in this section are more flexible than `URLSimpleDownload` (page 18) and `URLSimpleUpload` (page 21) and they are asynchronous, returning control to your application immediately.

Your application will use the following functions regardless of whether it is uploading or downloading data:

- `URLOpen` (page 27) to start the process of downloading data from a URL to a file or uploading data from file to a URL. Your application may then call the other functions described in this section to control the download or upload process.
- `URLAbort` (page 30) terminates an upload or download process that was started by calling `URLOpen`.

Your application may use the following functions when it is downloading data:

- `URLGetDataAvailable` (page 30) determines the amount of data that your application can retrieve from buffers.
- `URLGetBuffer` (page 32) retrieves the next buffer of data in download operation.
- `URLReleaseBuffer` (page 33) releases the buffer obtained by a call to `URLGetBuffer`.
- `URLReadBuffer` (page 34) copies data into a specified buffer.

URLOpen

Opens a URL and starts an asynchronous download or upload operation.

```
OSStatus URLOpen (
                    URLReference urlRef,
                    FSSpec* fileSpec,
                    URLOpenFlags openFlags,
                    URLNotifyProcPtr notifyProc,
                    URLEventMask eventRegister,
                    void* userContext);
```

`urlRef` A reference of type `URLReference` (page 46) that specifies the URL you want to open. Call `URLNewReference` (page 39) to create `urlRef`.

`fileSpec` A pointer to a structure of type `FSSpec` that identifies the file to which data is to be downloaded or from which data is to be uploaded, or `NULL`. The `fileSpec` parameter must be an `FSSpec` for upload operations.

When `fileSpec` is an `FSSpec`, `URLOpen` automatically starts and completes the transfer of data between the URL specified by the `urlRef` parameter and the specified file. Your application does not have access to buffer state and event information and cannot call `URLGetBuffer`, `URLReleaseBuffer`, `URLReadBuffer`, or `URLGetDataAvailable`.

If you specify `NULL` for a download operation, your application must call `URLReadBuffer` (page 34) or `URLGetBuffer` (page 32) to retrieve the data as it is downloaded.

When `fileSpec` is an `FSSpec` that specifies a file that exists and you want to replace it, specify the path to the destination file, terminate the path with a forward slash (/), and specify `kURLReplaceExistingFlag` in the `openFlags` parameter. If you specify a name that already exists on the server and you do not specify `kURLReplaceExistingFlag` in the `openFlags` parameter, `URLOpen` returns the error `kURLDestinationExistsError`. If you do not specify a name, do not specify `kURLReplaceExistingFlag`, and the name already exists on the server, the URL Access Manager creates a unique name by appending a number to the original name before the extension, if any.

For more information about the `FSSpec` structure, see *Inside Macintosh: Files*.

`openFlags` A value of type `URLOpenFlags` (page 3-47) that specifies data transfer options. The following constants can be used to specify download options:

```
kURLReplaceExistingFlag
kURLExpandFile Flag
kURLDisplayAuthFlag
kURLDoNotTryAnonymousFlag
```

To specify an upload operation, set `kURLUploadFlag` in `openFlags`. You may also want to set one or more of the following constants to specify upload options:

```
kURLReplaceExistingFlag
kURLBinHexFile Flag
kURLDisplayAuthFlag
kURLDoNotTryAnonymousFlag
```

See "URL Open Flag Constants" (page 47) for descriptions of these constants.

`notifyProc` A pointer to an application-defined event notification routine as described in "Notification Callback Routine" (page 43) or `NULL`. If you provide this parameter, your event notification routine is called each time one of the events specified in the `eventRegister` parameter occurs.

If your application does not provide an event notification routine, set `notifyProc` to `NULL`. To get status information you can periodically call `URLGetCurrentState` (page 40) to monitor the data transfer.

`eventRegister` A value of type `URLEventMask` that specifies the events for which your application-defined notification routine should be called. See "URL Event Constants" (page 51) for a description of the possible values.

`userContext` An untyped pointer to arbitrary data that the URL Access Manager will pass to your notification callback routine when it is called.

function result A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The `URLOpen` function starts an asynchronous download or upload operation and returns immediately. If the `fileSpec` parameter is a valid `FSSpec` structure, the URL Access Manager continues to transfer data until the transfer is complete.

To upload data, the `fileSpec` parameter must be an `FSSpec` structure.

To download data, the `fileSpec` parameter can be an `FSSpec` structure or `NULL`. If `fileSpec` is `NULL`, `URLOpen` starts the data transfer, but your application must call

URLGetBuffer (page 3-32) or URLReadBuffer (page 3-34) to complete the data transfer.

URLAbort

Terminates a data transfer.

```
OSStatus URLAbort (URLReference urlRef);
```

urlRef	A reference of type URLReference (page 46) that identifies the URL for which you want to terminate a data transfer.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The URLAbort function terminates a data transfer that was started by URLOpen. When your application calls URLAbort, the URL Access Manager changes the state returned by URLGetCurrentState (page 3-40) to kURLAbortingState. If you provided an event notification routine, the URL Access Manager calls it and passes kURLAbortInitiatedEvent to it.

When the data transfer is terminated, the URL Access Manager changes the state returned by URLGetCurrentState (page 3-40) to kURLCompletedState. If you provided an event notification routine, the URL Access Manager calls it and passes kURLCompletedEvent to it.

URLGetDataAvailable

Obtains the amount of data available for retrieval in a download operation.

```
OSStatus URLGetDataAvailable (
                    URLReference urlRef,
                    Size *dataSize);
```

`urlRef`	A reference of type `URLReference` (page 46) that identifies the URL whose buffer size you want.
`dataSize`	A pointer to a value of type `Size`. On return, `dataSize` is set to the number of bytes that can be retrieved. For more information on the `Size` type, see *Inside Macintosh: Memory.*
function result	A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The `URLGetDataAvailable` function obtains the amount of data in bytes that is available for your application to retrieve. This function returns meaningful data only if you have initiated a download process.

IMPORTANT

The `URLGetDataAvailable` function returns only the number of bytes in buffers that remain after you have already copied data from buffers by calling `URLGetBuffer` (page 32) or `URLReadBuffer` (page 34). The number returned does not include the number of bytes in transit to a buffer, nor does it include the amount of data not yet transferred from the URL. ▲

This function returns the amount of data remaining in buffers, so do not use it to calculate the amount of remaining data to be downloaded. To determine the amount of data remaining to be downloaded, call `URLGetProperty` (page 36) and specify the `kURLResourceSize` property in `URLCallbackInfo` (page 46). You can then subtract the amount of data copied by `URLGetBuffer` (page 32) or `URLReadBuffer` (page 34) to determine the amount of data remaining to be transferred to your application.

URLGetBuffer

Gets the next buffer of data in a download operation.

```
OSStatus URLGetBuffer (
                    URLReference urlRef,
                    void** buffer,
                    Size *bufferSize);
```

`urlRef`	A reference of type `URLReference` (page 46) that identifies the URL whose data is being downloaded.
`buffer`	An untyped pointer to a pointer to a buffer. On return, the buffer contains the downloaded data.
`bufferSize`	A pointer to a value of type `Size`. On return, `bufferSize` contains the number of bytes of data in the buffer. For more information on type `Size` see *Inside Macintosh: Memory.*
function result	A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The `URLGetBuffer` function obtains the next buffer of data in a download operation. The `URLGetBuffer` function is used by applications that call `URLOpen` with a `fileSpec` parameter that is `NULL` and that do not need to retain or modify the returned data.

Each buffer returned by `URLGetBuffer` is provided by the URL Access Manager, so your application should call `URLReleaseBuffer` (page 3-33) as soon as possible to prevent the URL Access Manager from running out of buffers.

You should call `URLGetBuffer` repeatedly until your notification callback routine returns `kURLCompletedEvent` or `kURLAbortInitiatedEvent`, or until `URLGetStatus` returns `kURLTransactionComplete` or `kURLAbortingState`. Between calls to `URLGetBuffer`, you should call `URLIdle` to allow time for the URL Access Manager to refill its buffers.

▲ **WARNING**
The data in the buffer returned by `URLGetBuffer` cannot be modified or retained for a long period of time. If your application needs to modify or retain the data in `buffer`, it should call `URLReadBuffer` (page 3-34) instead of calling `URLGetBuffer`. ▲

URLReleaseBuffer

Releases a buffer that belongs to the URL Access Manager.

```
OSStatus URLReleaseBuffer (
                    URLReference urlRef,
                    void* buffer);
```

`urlRef`	A reference of type `URLReference` (page 46) that identifies the URL for which you want to release a buffer.
`buffer`	An untyped pointer to the buffer you want to release.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The `URLReleaseBuffer` function releases a buffer obtained by calling `URLGetBuffer` (page 32). To prevent the URL Access Manager from running out of buffers, you should call `URLReleaseBuffer` as soon as possible after calling `URLGetBuffer` (page 32).

Do not call `URLReleaseBuffer` to dispose of buffers that your application allocated for use with the `URLReadBuffer` (page 34) function. Such buffers are your own and you should reuse or dispose of them yourself.

URLReadBuffer

Copies the next buffer of data in a download operation.

```
OSStatus URLReadBuffer (
                    URLReference urlRef,
                    void* buffer,
                    Size requestedSize,
                    Size *actualSize);
```

`urlRef`	A reference of type `URLReference` (page 46) that identifies the URL whose data is being downloaded.
`buffer`	An untyped pointer to a buffer for receiving the data. On return, this buffer contains data from the specified URL. Your application is responsible for allocating `buffer` and disposing of it when it is no longer needed.
`requestedSize`	A value of type `Size`. Before calling `URLReadBuffer`, set this value to the number of bytes you want to copy. The `URLReadBuffer` function copies all of the data not yet retrieved into `buffer`, up to the requested size. If `URLReadBuffer` does not fill `buffer`, it does not wait for more data to become available.
`actualSize`	A pointer to value of type `Size`. On return, this value is the actual number of bytes copied. For more information on type `Size`, see *Inside Macintosh: Memory*.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The `URLReadBuffer` function copies the next buffer of data in a download operation into your own buffer. The `URLReadBuffer` function is used by applications that call `URLOpen` with a `fileSpec` parameter that is `NULL` and that need to retain or modify the returned data.

You should call `URLReadBuffer` repeatedly until your notification callback routine returns `kURLCompletedEvent` or `kURLAbortInitiatedEvent` or until `URLGetStatus` returns `kURLTransactionComplete` or `kURLAbortingState`. Between

calls to `URLReadBuffer`, you should call `URLIdle` to allow time for the URL Access Manager to refill its buffers.

You are responsible for allocating `buffer` and deallocating it when it is no longer needed; do not call `URLReleaseBuffer` (page 33) to dispose of `buffer`.

SEE ALSO

`URLGetBuffer` (page 32).

Getting and Setting URL Properties

You can use these utility functions to set or retrieve information about a URL or the resource the URL points to:

- `URLGetPropertySize` (page 35) gets the size of a property that can be retrieved by `URLGetProperty`.
- `URLGetProperty` (page 36) retrieves a property associated with a URL.
- `URLSetProperty` (page 37) sets a property value in a URL.

The value of a property may change during an upload or download. When a change occurs during a data transfer, the URL Access Manager calls your notification callback routine with an event code of `kURLPropertyChangedEvent`.

For a list of the universal properties that are defined and reserved by Apple Computer, Inc., see "Universal Properties" (page 56). For a list of HTTP-specific properties that are defined and reserved by Apple Computer, Inc., see "HTTP and HTTPS Properties" (page 58).

URLGetPropertySize

Gets the size of a property.

```
OSStatus URLGetPropertySize (
                    URLReference urlRef,
                    const char* property,
                    Size *propertySize);
```

`urlRef`	A reference of type `URLReference` that identifies the URL that has a property whose size you want to obtain.
`property`	A pointer to a null-terminated array of characters that specifies the name of the property whose size you want to obtain.
`propertySize`	A pointer to a value of type `Size`. On return, `propertySize` contains the size of the property you want to obtain or -1 if the size is not available. For more information on the `Size` type, see *Inside Macintosh: Memory*.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The `URLGetPropertySize` function obtains the size of a property in bytes. You should call `URLGetPropertySize` before calling `URLGetProperty` (page 3-36) to make sure the buffer you use when you call `URLGetProperty` is big enough to hold the property value. Pass the value returned in `propertySize` as a parameter to the `URLGetProperty` function.

URLGetProperty

Gets the value of a property.

```
OSStatus URLGetProperty (
                    URLReference urlRef,
                    const char* property,
                    void* propertyBuffer,
                    Size bufferSize);
```

`urlRef`	A reference of type `URLReference` that identifies the URL that has a property whose value you want to obtain.
`property`	An untyped pointer to a null-terminated array of characters that specifies the name of the property whose value you want to obtain.

`propertyBuffer`
A pointer to a buffer. On return, `buffer` contains the value of the specified property.

`bufferSize` A value of type `Size` that specifies the length of `propertyBuffer` in bytes. Before calling `URLGetProperty`, call `URLGetPropertySize` to make sure that `propertyBuffer` is big enough to hold the value that will be returned by `URLGetProperty`. For more information on the `Size` type, see *Inside Macintosh: Memory*.

function result A result code. If `URLGetProperty` returns `kURLPropertyBufferTooSmallError`, your application should allocate a new buffer of the length returned by `URLGetPropertySize` calling `URLGetProperty` again. For a list of other possible result codes, see "Result Codes" (page 59).

DISCUSSION

Before calling `URLGetProperty`, call `URLGetPropertySize` (page 35) to make sure that your buffer is big enough to hold the property value. Then specify the size of the buffer required for that property value in the `bufferSize` parameter.

SEE ALSO

The function `URLSetProperty` (page 37).

URLSetProperty

Sets the value of a property.

```
OSStatus URLSetProperty (
                    URLReference urlRef,
                    const char* property,
                    void* propertyBuffer,
                    Size bufferSize);
```

`urlRef` A reference of type `URLReference` that identifies the URL that has a property whose value you want to set.

`property`	An untyped pointer to a null-terminated array of characters that specifies the name of the property you want to set. See "URL Property Name Constants" (page 56) for the list of properties you can set.
`propertyBuffer`	A pointer to a buffer containing the value you want to set.
`bufferSize`	A value of type `Size` that specifies the size of the buffer pointed to by `propertyBuffer`. For more information on the `Size` type, see *Inside Macintosh: Memory.*
function result	A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The `URLSetProperty` function sets the value of a property that is associated with a URL. To clear the value of a property, set it with an empty character array.

The following properties defined by Apple Computer, Inc., can be set:

```
kURLFileType
kURLFileCreator,
kURLUserName
kURLPassword
kURLHTTPRequestMethod
kURLHTTPRequestHeader
kURLHTTPRequestBody
kURLHTTPUserAgent
```

URL Access Manager Utility Functions

You can use these utility functions to create and dispose of URL references, retrieve the state of a URL that has been opened by `URLOpen`, determine an error condition, yield time so that the URL Access Manager can refill its buffers, or get information about a file.

- `URLNewReference` (page 39) creates a reference to a URL.
- `URLDisposeReference` (page 39) disposes of memory used by a URL reference.
- `URLGetCurrentState` (page 40) returns the state of the specified URL.

- `URLGetError` (page 41) retrieves the error that caused a download or upload operation to fail.
- `URLIdle` (page 42) gives the URL Access Manager time to refill its buffers.
- `URLGetFileInfo` (page 42) returns a Macintosh file type and creator as specified by the Internet configuration mapping table.

URLNewReference

Creates a reference to a URL.

```
OSStatus URLNewReference (
                    const char* url,
                    URLReference *urlRef);
```

`url`	A pointer to an array of characters that specify the URL you want to reference.
`urlRef`	A pointer to a structure of type `URLReference` (page 46). On return, `urlRef` points to a valid URL reference that you can use as a parameter to many URL Access Manager functions.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The function `URLNewReference` creates a reference to a URL that you can use in subsequent calls to the URL Access Manager.

When you no longer need a URL reference, you should reclaim memory by calling `URLDisposeReference` (page 39).

URLDisposeReference

Releases memory used by a URL reference.

```
OSStatus URLDisposeReference  (URLReference urlRef);
```

`urlRef`	A reference of type `URLReference` (page 46) that identifies the URL you want to dispose of.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The `URLDisposeReference` function releases memory occupied by a URL reference. You should `URLDisposeReference` when you no longer need a URL reference.

SEE ALSO

The function `URLNewReference` (page 39).

URLGetCurrentState

Gets the current state of a URL.

```
OSStatus URLGetCurrentState (
            URLReference urlRef,
            URLState* state);
```

`urlRef`	A reference of type `URLReference` that identifies the URL whose state information you want to obtain.
`state`	A pointer to a value of type `URLState`. On return, `state` contains the current state of the URL.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The `URLGetCurrentState` function obtains the current state of a data transfer with respect to the specified URL.

The following state constants can be returned at any time:

```
kURLNullState
kURLInitiatedState
kURLResourceFoundState
kURLDownloadingState
kURLAbortingState
kURLTransactionCompleteState
kURLErrorOccurredState
kURLUploadingState
```

The following state constants can be returned if the `fileSpec` parameter was `NULL` when your application called `URLOpen` (page 27):

```
kURLDataAvailableState,
kURLTransactionCompleteState
```

For a description of these constants, see "URL State Constants" (page 49).

URLGetError

Gets the error code that caused a download or upload operation to fail.

```
OSStatus URLGetError (
                    URLReference urlRef,
                    OSStatus* urlError);
```

`urlRef`	A reference of type `URLReference` that identifies the URL for which the error code is to be obtained.
`urlError`	A pointer to a value of type `OSStatus`. On return, `urlError` contains the first nontrivial error the URL encountered. A nontrivial error is an error that cannot be recovered from and that caused the download or upload to fail.
function result	A result code or a protocol-specific error code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The `URLGetError` function obtains the error code that caused a download or upload operation to fail. The error code may be a system error code, a

protocol-specific error code, or one of the error codes listed in "Result Codes" (page 59).

URLIdle

Gives the URL Access Manager time to refill its buffers.

```
OSStatus URLIdle (void);
```

function result A result code. For a list of possible result codes, see"Result Codes" (page 59).

DISCUSSION

The `URLIdle` function gives the URL Access Manager time to refill its buffers during download operations. Your application must call `URLIdle` from its main event loop if it calls `URLGetBuffer` or `URLReadBuffer` to download data from a URL.

URLGetFileInfo

Obtains the file type and creator for a file name.

```
OSStatus URLGetFileInfo (
                    StringPtr name,
                    OSType* type,
                    OSType* creator);
```

`name` A pointer to a Pascal string. Before calling `URLGetFileInfo`, set the string to the name of the file for which you want information.

`type` A pointer to a value of type `OSType`. On return, this value identifies the file type of the file.

`creator` A pointer to a value of type `OSType`. On return, this value identifies the file creator.

function result A result code. For a list of possible result codes, see "Result Codes" (page 59).

DISCUSSION

The `URLGetFileInfo` function obtains a file's type and creator codes. The type and creator returned in `type` and `creator` are determined by the Internet configuration mapping table and are based on the filename extension. For example, if you pass the file name `jane.txt`, `URLGetFileInfo` will return `'TEXT'` in the `type` parameter and `'ttxt'` in the `creator` parameter.

URL Access Manager Application-Defined Routines

This section describes two types of application-defined routines that your application can provide:

- A notification callback routine that receives events when you transfer data asynchronously by calling `URLOpen` (page 27).
- A system event callback routine that receives system events when you transfer data synchronously by calling `URLSimpleDownload` (page 18), `URLSimpleUpload` (page 21), `URLDownload` (page 23), or `URLUpload` (page 25).

Notification Callback Routine

When you call `URLOpen` (page 27), you may want to specify a notification callback routine to receive information about events that occur during an asynchronous download or upload. This is how you would declare the callback function if you were to name the function `MyURLNotifyProc`:

```
OSStatus *MyURLNotifyProcPtr (
                    void userContext,
                    URLEvent event,
                    URLCallbackInfo *callbackInfo);
```

`userContext`	An application-defined value that your application previously passed as a parameter when it called `URLOpen` (page 27). When an event occurs, the URL Access Manager passes `userContext` back to you.
`event`	A value of type `URLEvent` specifying the event that triggered the callback. The type of event that can trigger a callback depends on whether you called `URLOpen` with a `fileSpec` that is a valid `FSSpec`. See the discussion section for details.
`callbackInfo`	A pointer to a structure of type `URLCallbackInfo` (page 46). On return, this structure contains information relevant to the event that has occurred.
result	Your notification callback routine should always return `noErr`.

DISCUSSION

The URL Access Manager will call your notification callback routine when events for which your application has registered occur during the asynchronous download or upload of data to a URL. Use the `eventRegister` parameter to the `URLOpen` (page 27) function to specify the events for which you want to receive notification.

If you call `URLOpen` with a `fileSpec` parameter that is a valid `FSSpec`, the following events can cause the URL Access Manager to call your application's notification callback routine:

```
kURLPercentEvent
kURLPeriodicEvent
kURLPropertyChangedEvent
kURLSystemEvent
kURLInitiatedEvent
kURLResourceFoundEvent
kURLDownloadingEvent
kURLUploadingEvent
kURLAbortInitiatedEvent
kURLCompletedEvent
kURLErrorOccurredEvent
```

If you call `URLOpen` with a `fileSpec` parameter that is `NULL` and you are downloading from a URL, the following events can cause the URL Access Manager to call your application's notification callback routine:

```
kURLDataAvailableEvent
kURLTransactionCompleteEvent
```

See "URL Event Constants" (page 51) for a description of each constant.

System Event Callback Routine

If your application calls `URLSimpleDownload` (page 18) or `URLSimpleUpload` (page 21), you may want to provide a routine that receives system events that may occur while the URL Access Manager displays a dialog box with a progress indicator or an authentication dialog box. A typical system event callback routine might look like this:

```
void MyURLSystemEventCBPRoc(
                    void* userContext,
                    URLReference urlRef,
                    EventRecord *event);
```

`userContext` An untyped pointer to arbitrary data that your application previously passed to `URLSimpleDownload` (page 18) or `URLSimpleUpload` (page 21).

`urlRef` A reference of type `URLReference` that identifies the URL that caused the event.

`event` A pointer to a structure of type `eventRecord` that describes the event that triggered the callback. For more information on the `EventRecord` structure see *Inside Macintosh: Overview*.

result Your system event callback routine should process the system event and return `noErr`.

DISCUSSION

If your application asks the URL Access Manager to display a progress indicator or an authentication dialog box during a call to `URLSimpleDownload` (page 18), `URLSimpleUpload` (page 21), `URLDownload` (page 23), or `URLUpload` (page 25), your application should provide a system event callback routine to handle events that occur while the progress indicator or authentication dialog box is being displayed.

If your application does not provide a system event callback routine, the URL Access Manager uses a non-movable modal dialog box to display the progress indicator or authentication dialog box.

When your system event callback routine is called, it should process the event immediately.

URL Access Manager Structures and Other Data Types

The following structures and other data types supply the information you need to use URL Access Manager functions:

- The data type `URLReference` (page 46) is a reference to a Universal Resource Locator (URL).
- The structure `URLCallbackInfo` (page 46) returns information required by your notification callback routine (page 3-43).

URLReference

The `URLReference` data type is an opaque data structure that refers to a URL. Most URL Access Manager functions require a `URLReference`, which is defined as follows:

```
typedef struct OpaqueURLReference* URLReference;
```

You call the function `URLNewReference` (page 39) to create a URL reference. You call the function `URLDisposeReference` (page 39) to dispose of the reference when you no longer need it.

URLCallbackInfo

If your application defines a notification routine, the URL Access Manager returns event information to your application in a data structure of type `URLCallbackInfo`.

```
struct URLCallbackInfo{
    UInt32          version;
    URLReference    urlRef;
    const char*     property;
    UInt32          currentSize;
    EventRecord*    systemEvent;
};
typedef struct URLCallbackInfo URLCallbackInfo;
```

Field descriptions

`version`	The version of the `URLCallbackInfo` structure.
`urlRef`	A reference to the URL associated with the event.
`property`	A pointer to a character constant that specifies predefined properties. See "URL Property Name Constants" (page 56) for detailed description of these constants.
`currentSize`	The size of the buffer containing the property value.
`systemEvent`	A pointer to the system event record (used for system events only).

URL Constants

The following sections describe the URL Access Manager constants:

- "URL Open Flag Constants" (page 47)
- "URL State Constants" (page 49)
- "URL Event Constants" (page 51)
- "URL Event Mask Constants" (page 53)
- "URL Property Name Constants" (page 56)
- "URL Authentication Flag Constant" (page 59)

URL Open Flag Constants

The URL Access Manager defines constants that you can use to specify options for downloading data from a URL or for uploading data to a URL. The

constants for specifying these options are defined in the `URLOpenFlags` enumeration.

```
enum
{
    kURLReplaceExistingFlag         = 1 << 0,
    URLRBinHexFileFlag              = 1 << 1,
    kURLExpandFileFlag              = 1 << 2,
    kURLDisplayProgressFlag         = 1 << 3,
    kURLDisplayAuthFlag             = 1 << 4,
    KURLUploadFlag                  = 1 << 5,
    kURLIsDirectoryHintFlag         = 1 << 6,
    kURLDoNotTryAnonymousFlag       = 1 << 7,
    kURLDirectoryListingFlag        = 1 << 8,
    };
typedef UInt32 URLOpenFlags;
```

Constant Descriptions

`kURLReplaceExistingFileFlag`
If the bit accessed by this flag is set and the specified file or directory already exists, the contents of the specified file or directory are replaced by the newly downloaded or uploaded data. If this bit is not set, the name of the file or directory isn't specified, and the file or directory already exists, a number is appended to the name before any extension until a unique name is created, and the data is downloaded or uploaded to the new file or directory name without notifying the calling application that the name has changed. In the case of a download operation, your application can check the `FSSpec` to obtain the new file name.

`kURLBinHexFileFlag` If the bit accessed by this flag is set, the URL Access Manager converts the file to BinHex format before it uploads a file that is not of type `'TEXT'` and that has a resource fork.

`kURLExpandFileFlag`
If the bit accessed by this flag is set, files with a `.bin` or `.hqx` extension are decoded. If the Stuffit Engine is installed in the System Folder, it is used to decompress the data.

`kURLDisplayProgressFlag`
: If the bit accessed by this flag is set, `URLSimpleDownload` (page 18), `URLSimpleUpload` (page 21), `URLDownload` (page 23), and `URLUpload` (page 25) display a progress indicator in a modal dialog box during the download or upload operation.

`kURLDisplayAuthFlag`
: If the bit accessed by this flag is set, a modal authentication dialog box is displayed if authentication is required.

`kURLUploadFlag`
: If the bit accessed by this flag is set, `URLOpen` uploads data to the specified URL.

`kURLIsDirectoryHintFlag`
: If the bit accessed by this flag is set, the URL Access Manager assumes that the URL points to a directory (for download operations only).

`kURLDoNotTryAnonymousFlag`
: If the bit accessed by this flag is set and if `kURLDisplayAuth` is set, `URLDownload`, `URLUpload`, `URLSimpleDownload` and `URLSimpleUpload` do not try to log on to FTP servers anonymously. Instead, they immediately display an authentication dialog box. If this bit is not set, `URLDownload`, `URLUpload`, `URLSimpleDownload` and `URLSimpleUpload` first attempt to log on to FTP servers anonymously.

`kURLDirectoryListingFlag`
: If the bit accessed by this flag is set, a listing of the directory is downloaded instead of the entire directory. If the URL points to a file instead of a directory, the file is downloaded.

URL State Constants

URL state constants are returned by the function `URLGetCurrentState` (page 40). The constants are defined in the `URLState` enumeration.

```
enum
{
    kURLNullState               = 0,
    kURLInitiatedState          = 1,
    reserved1                   = 2,
```

```
    reserved2                   = 3,
    kURLResourceFoundState      = 4,
    kURLDownloadingState        = 5,
    kURLDataAvailableState      = 0x10 + kURLDownloadingState,
    kURLTransactionCompleteState= 6,
    kURLErrorOccurredState      = 7,
    kURLAbortingState           = 8,
    kURLCompletedState          = 9,
    kURLUploadingState          = 10
    };

typedef UInt32 URLState;
```

Constant Descriptions

`kURLNullState` The function `URLOpen` (page 27) has not yet been called.

`kURLInitiatingState`
The function `URLOpen` (page 27) has been called; however, the location specified by the URL reference has not yet been accessed. The stream enters this state from the `kURLNullState` state.

`reserved1` Reserved.

`reserved2` Reserved.

`kURLResourceFoundState`
The location specified by the URL reference has been accessed and is valid. The stream enters this state from the `kURLInitiatingState` state.

`kURLDownloadingState`
The download operation is in progress but there is currently no data in the buffers. The stream enters this state initially from the `kURLResourceFoundState` state. During a download operation, the stream's state may alternate between the `kURLDownloadingState` and the `kURLDataAvailableState` states.

`kURLDataAvailableState`
The download operation is in progress and there is data available in buffers. The stream enters this state initially from the `kURLDownloadingState` state. During a download operation, the stream's state may alternate between the

kURLDownloadingState and the kURLDataAvailableState states.

kURLTransactionCompleteState
The download or upload operation is complete. The stream can enter this state from the kURLDownloadingState state.

kURLErrorOccurredState
An error occurred while transferring data. The stream can enter this state from any state except the kURLAbortingState state.

kURLAbortingState The download or upload operation is in progress. You enter this state from the kURLErrorOccurredState state, or by calling the URLAbort function URLAbort (page 30) while the stream is in any other state.

kURLCompletedState
There is no more activity to be performed on this stream. The transferred has completed successfully, the transfer has been aborted, or an error has occurred. The stream enters this state from the kURLTransactionCompleteState or the kURLAbortingState state.

kURLUploadingState The upload operation is in progress.

URL Event Constants

URL event constants are passed to your notification callback routine (page 3-43) to receive the type of event that occurred. The URL event constants are defined in the URLEvent enumerator.

```
enum
{
    kURLInitiatedEvent              = kURLInitiatingState,
    kURLResourceFoundEvent          = kURLResourceFoundState,
    kURLDownloadingEvent            = kURLDownloadingState,
    kURLAbortInitiatedEvent         = kURLAbortingState,
    kURLCompletedEvent              = kURLCompletedState,
    kURLErrorOccurredEvent          = kURLErrorOccurredState,
    KURLDataAvailableEvent          = kURLDataAvailableState,
    kURLTransactionCompleteEvent    = kURLTransactionCompleteState,
    kURLUploadingEvent              = kURLUploadingState,
```

```
    kURLSystemEvent                 = 29,
    kURLPercentEvent                = 30,
    kURLPeriodicEvent               = 31,
    kURLPropertyChangedEvent        = 32,
    };

typedef UInt32 URLEvent;
```

Constant Descriptions

`kURLInitiatedEvent`
: The function `URLOpen` (page 27) has been called; however, the location specified by the URL reference has not yet been accessed.

`kURLResourceFoundEvent`
: The location specified by the URL reference has been accessed and is valid.

`kURLDownloadingEvent`
: The download operation is in progress.

`kURLAbortInitiatedEvent`
: The download or upload operation has been aborted.

`kURLCompletedEvent`
: All operations associated with the function `URLOpen` (page 27) have been completed. This event indicates either the successful completion of a download or upload operation or the completion of cleanup work after aborting the download or upload operation.

`kURLErrorOccurredEvent`
: An error occurred.

`kURLDataAvailableEvent`
: Data is available in buffers.

`kURLTransactionCompleteEvent`
: The download operation is complete because there is no more data to retrieve from buffers.

`kURLUploadingEvent`
: The upload operation is in progress.

`kURLSystemEvent` A system event occurred.

`kURLPercentEvent` An increment of one percent of the data was transferred into buffers. This event occurs only when the size of the data being downloaded is known.

`kURLPeriodicEvent` A time interval of approximately one quarter of a second has passed.

`kURLPropertyChangedEvent`
A property, such as a filename or user name, becomes known or changes. For information about the properties that may cause this event to occur, see "URL Property Name Constants" (page 56).

URL Event Mask Constants

The URL Access Manager defines masks you can use to specify the events for which your notification callback routine should be called. You pass these constants to the function `URLOpen` (page 27) in the `eventRegister` parameter. The `URLEventMask` enumerator defines these mask constants.

```
enum
{
    kURLInitiatedEventMask      = 1 << (kURLInitiatedEvent - 1),
    kURLResourceFoundEventMask  = 1 << (kURLResourceFoundEvent - 1),
    kURLDownloadingMask         = 1 << (kURLDownloadingEvent - 1),
    kURLUploadingMask           = 1 << (kURLUploadingEvent - 1),
    kURLAbortInitiatedMask      = 1 << (kURLAbortInitiatedEvent - 1),
    kURLCompletedEventMask      = 1 << (kURLCompletedEvent - 1),
    kURLErrorOccurredEventMask  = 1 << (kURLErrorOccurredEvent - 1),
    kURLDataAvailableEventMask  = 1 << (kURLDataAvailableEvent - 1),
    kURLTransactionCompleteEventMask
                                = 1 << (kURLTransactionCompleteEvent -1),
    kURLSystemEventMask         = 1 << (kURLSystemEvent - 1 ),
    kURLPercentEventMask        = 1 << (kURLPercentEventMask -1 ),
    kURLPeriodicEventMask       = 1 << (kURLPeriodicEvent -1 ),
    kURLPropertyChangedEventMask
                                = 1 << (kURLPropertyChangedEvent -1 ),
    kURLAllBufferEventsMask     = kURLDataAvailableEventMask
                                        + kURLTransactionCompleteMask,
    kURLAllNonBufferEventsMask  = kURLinitiatedEventMask
                                        + kURLDownloadingMask
```

```
                                        + kURLUploadingMask
                                        + kURLAbortInitiatedMask
                                        + kURLCompletedEventMask
                                        + kURLErrorOccurredEventMask
                                        + kURLPercentEventMask
                                        + kURLPeriodicEventMask
                                        + kURLPropertyChangedEventMask,
    kURLAllEventsMask           = 0xFFFFFFFF
};
typedef UInt32 URLEventMask;
```

Constant Descriptions

`kURLInitiatedEventMask`
: Set the bit specified by this mask if you want to be notified when the function `URLOpen` (page 27) has been called but the location specified by the URL reference has not yet been accessed.

`kURLResourceFoundEventMask`
: Set the bit specified by this mask if you want to be notified when the location specified by the URL reference has been accessed and is valid.

`kURLDownloadingMask`
: Set the bit specified by this mask when you want to be notified that the download operation is in progress.

`kURLUploadingMask`
: Set the bit specified by this mask when you want to be notified that the upload operation is in progress.

`kURLAbortInitiatedMask`
: Set the bit specified by this mask when you want to be notified that the download or upload operation has been aborted.

`kURLCompletedEventMask`
: Set the bit specified by this mask when you want to be notified that all operations associated with the call to the function `URLOpen` (page 27) have been completed. This event indicates either the successful completion of an operation or the completion of cleanup work after aborting the operation.

`kURLErrorOccurredEventMask`
Set the bit specified by this mask when you want to be notified that an error has occurred.

`kURLDataAvailableEventMask`
Set the bit specified by this mask when you want to be notified that data is available in buffers. If you include a file specification when you call the function `URLOpen` (page 27), your notification callback routine is not called.

`kURLTransactionCompleteEventMask`
Set the bit specified by this mask when you want to be notified that the operation is complete because there is no more data to retrieve from buffers. If you specify an `FSSpec` when you call the `URLOpen` (page 27), your notification callback routine is not called.

`kURLPercentEventMask`
Set the bit specified by this mask when you want to be notified that an increment of one percent of the data has been transferred into buffers. This information is useful if your application displays a progress indictor. This event occurs only when the size of the data being transferred is known.

`kURLPeriodicEventMask`
Set the bit specified by this mask when you want to be notified that a time interval of approximately one quarter of a second has passed. You can use this event to report the progress of the download operation when the size of the data is unknown or for other processing that you want to do at a regular interval.

`kURLPropertyChangedEventMask`
Set the bit specified by this mask when you want to be notified that a property, such as a filename or user name, has become known or changes. For information about the properties that may cause this event to occur, see `URL Property Name Constants` (page 56).

`kURLAllBufferEventsMask`
Set the bit specified by this mask when you want to be notified that a buffer-related event occurred, specifically the `kURLDataAvailableEvent` event and `kURLTransactionCompleteEvent` event. If you include a file

specification when you call the function `URLOpen` (page 27), your notification callback routine is not called for buffer-related events.

`kURLAllNonBufferEventsMask`
Set the bit specified by this mask when you want to be notified that an event unrelated to a buffer occurred; these events include all events except the `kURLDataAvailableEvent` event and `kURLTransactionCompleteEvent` event.

`kURLAllEventsMask` The bit accessed by this mask indicates that an event of any kind occurs. If you include a file specification when you call the `URLOpen` (page 27) function, your notification callback function will not be called for the `kURLDataAvailableEvent` event and `kURLTransactionCompleteEvent` event.

URL Property Name Constants

These constants define properties that you set with the function `URLSetProperty` (page 37). Before you call `URLGetProperty` (page 36) to get the value of a property, you must call the function `URLGetPropertySize` (page 35) to obtain the size of the property.

All of the property name constants are defined as character constants.

There are three types of property name constants: universal properties, HTTP and HTTPS properties, and authentication type flags.

Universal Properties

```
const char* kURLURL                = “URLString”;
const char* kURLResourceSize       = “URLResourceSize”;
const char* kURLLastModifiedTime   = “URLLastModifiedTime”;
const char* kURLMIMEType           = “URLMIMEType”;
const char* kURLFileType           = “URLFileType”;
const char* kURLFileCreator        = “URLFileCreator”;
const char* kURLCharacterSet       = “URLCharacterSet”;
const char* kURLResourceName       = “URLResourceName”;
const char* kURLHost               = “URLHost”;
```

```
const char* kURLAuthType            = "URLAuthType";
const char* kURLUserName            = "URLUserName";
const char* kURLPassword            = "URLPassword";
const char* kURLStatusString        = "URLStatusString";
const char* kURLIsSecure            = "URLIsSecure";
const char* kURLCertificate         = "URLCertificate";
const char* kURLTotalITems          = "URLTotalItems";
```

Constant Descriptions

`kURLURL` The URL name string.

`kURLResourceSize` The total size of the data at the location specified by the URL.

`kURLLastModifiedTime` The last time the data was modified.

`kURLMIMEType` The MIME type of the data.

`kURLFileType` The file type as specified in the call to the function `URLOpen` (page 27). If the call to the function `URLOpen` (page 27) did not specify a file, `kURLFileType` returns a file type compatible with the MIME type.

`kURLFileCreator` The file creator as specified in the call to the `URLOpen` (page 27) function. If the call to the `URLOpen` function did not specify a file, `kURLFileCreator` returns a creator that is compatible with the MIME type.

`kURLCharacterSet` The character set that the URL uses as returned by the HTTP server.

`kURLResourceName` The name associated with the data to be downloaded.

`kURLHost` The host on which the data is located.

`kURLAuthType` The type of authentication that the download operation requires. The default authentication type is `kUserNameAndPasswordFlag`.

`kURLUserName` The user name used for authentication. You can set this property.

`kURLPassword` The password used for authentication. You can set this property.

`kURLStatusString` The string that contains the stream's current status. You can use this property to display the status.

kURLIsSecure	A Boolean variable, specifying whether or not the download operation is secure.
kURLCertificate	The certificate provided by a remote server.
kURLTotalItems	The total number of items being uploaded or downloaded.

HTTP and HTTPS Properties

If you are using HTTP or HTTPS, you can use the following constants to specify additional properties when you call URLSetProperty and URLGetPropertySize, and URLGetProperty:

```
const char* kURLHTTPRequestMethod   = "URLHTTPRequestMethod";
const char* kURLHTTPRequestHeader   = "URLHTTPRequestHeader";
const char* kURLHTTPRequestBody     = "URLHTTPRequestBody";
const char* kURLHTTPRespHeader      = "URLHTTPRespHeader";
const char* kURLHTTPUserAgent       = "URLHTTPUserAgent";
```

Constant Descriptions

kURLHTTPRequestMethod

The HTTP method to be used in the request. You should call URLGetPropertySize (page 35) to retrieve the size of the request method before you retrieve the property itself. If you are posting a form, you must set this property.

kURLHTTPRequestHeader

The HTTP request header. You should call URLGetPropertySize (page 35) to retrieve the size of the request header before you retrieve the property itself. You may set this property to contain all headers needed for the request. If you are posting a form and have set the kURLHTTPRequestMethod and kURLHTTPRequestBody properties, you do not need to set this property.

kURLHTTPRequestBody

The HTTP request body to be provided in the request. You should call URLGetPropertySize (page 35) to retrieve the size of the request body before you retrieve the property itself. If you set this property but not the kURLHTTPHeader property, a body-length header is automatically added to the request. If you are posting a form, you must set this property.

`kURLHTTPRespHeader` The HTTP response header. You should call `URLGetPropertySize` (page 35) to retrieve the size of the response header before you retrieve the property itself.

`kURLHTTPUserAgent` The HTTP user agent string that is embedded in HTTP requests. By default, the URL Access Manager sets the user agent string to "URL Access 1.0 (Macintosh ; PPC)". You can set this property.

URL Authentication Flag Constant

The `kUserNameAndPasswordFlag` authentication flag specifies that both the user name and password are used for authentication.

```
enum
{
    kUserNameAndPasswordFlag    = 0x00000001
};
```

Result Codes

The result codes specific to the URL Access Manager are listed here. Note that some errors, such as system errors, do not appear in this list.

`kURLNoErr`	0	No error
`kURLInvalidURLReferenceError`	–30770	Invalid URL reference
`kURLProgressAlreadyDisplayedError`	–30771	A dialog box with a progress indicator already displayed
`kURLDestinationExistsError`	–30772	Destination file already exists
`kURLInvalidURLError`	–30773	Invalid URL format
`kURLUnsupportedSchemeError`	–30774	Transfer protocol is not supported
`kURLServerBusyError`	–30775	Server is busy
`kURLAuthenticationError`	–30776	Server identification has failed
`kURLPropertyNotYetKnownError`	–30777	The value of the property is not available yet

kURLUnknownPropertyError	–30778	Invalid or undefined property
kURLPropertyBufferTooSmallError	–30779	Buffer too small to receive the requested property
kURLUnsettablePropertyError	–30780	Cannot set property
kURLInvalidCallError	–30781	Invalid call
kURLFileEmptyError	–30783	Resource file empty
kURLExtensionFailureError	–30785	Extension fails to load
kURLInvalidConfigurationError	–30786	Invalid configuration
kURLAccessNotAvailableError	–30787	The URL Access Manager is not available