INSIDE MACINTOSH

Transferring Data With the URL Access Manager

April 21, 1999
Technical Publications
© 1999 Apple Computer, Inc.

Apple Computer, Inc.

© 1999 Apple Computer, Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc., except to make a backup copy of any documentation provided on CD-ROM.

The Apple logo is a trademark of Apple Computer, Inc.
Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this book. Apple retains all intellectual property rights associated with the technology described in this book. This book is intended to assist application developers to develop applications only for Apple-labeled or Apple-licensed computers.

Every effort has been made to ensure that the information in this manual is accurate. Apple is not responsible for typographical errors.

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, and Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Adobe, Acrobat, and PostScript are trademarks of Adobe Systems Incorporated or its subsidiaries and may be registered in certain jurisdictions.

Helvetica and Palatino are registered trademarks of Linotype-Hell AG and / or its subsidiaries.

ITC Zapf Dingbats is a registered trademark of International Typeface Corporation.

QuickView™ is licensed from Altura Software, Inc.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

© Apple Computer, Inc. 4/2/99

Contents

Figures, Tables, and Listings

© Apple Computer, Inc. 4/21/99

© Apple Computer, Inc. 4/21/99

About This Manual

This manual describes the URL Access Manager, which you can use to transfer data between your application and Hypertext Transfer Protocol (HTTP) and File Transfer Protocol (FTP) Universal Resource Locators (URLs). You can download data from a URL to a file, a directory, or a location in memory. You can upload data from a file or directory to an FTP URL using an anonymous FTP session or an authenticated FTP session. For the purpose of testing your application on a computer that does not have access to a HTTP or FTP server, you can also upload data from or download data to a URL that begins with file:///.

Conventions Used in This Manual

The Courier font is used to indicate text that you type or see displayed. This manual includes special text elements to highlight important or supplemental information:

Note
Text set off in this manner presents sidelights or interesting points of information. ◆

IMPORTANT
Text set off in this manner—with the word Important—presents important information or instructions. ▲

▲ **WARNING**
Text set off in this manner—with the word Warning—indicates potentially serious problems. ▲

© Apple Computer, Inc. 4/21/99

For more information

The following sources provide additional information that may be of interest to developers who use URL Access:

- *Network Services Location Manager SDK* , which describes a technology for locating URLs.
- *HTML The Definitive Guide* by Chuck Musciano and Bill Kennedy, O'Reilly & Associates, Inc. 1996.

© Apple Computer, Inc. 4/21/99

About the URL Access Manager

The URL Access Manager provides routines that allow your application to download data from or upload data to a Universal Resource Locator (URL) using four protocols:

- Hypertext Transfer Protocol (HTTP) and secure Hypertext Transfer Protocol (HTTPS). These protocols allow you to
  - download data with optional 40-bit RSA encryption
  - send HTML form information to a URL using the POST and the GET methods
  - obtain information about URLs; this information is known as URL properties
  - set certain URL properties
- File Transfer Protocol (FTP). Using FTP, you can
  - download files and directories
  - upload files and directories
  - obtain URL properties
  - set certain URL properties

  The URL Access Manager supports passive FTP connections (in which the client opens the data cconnection to the server) and active FTP connections (in which the server opens the data connection to the client).
- A file protocol. With the file:/ / / protocol, you can download files and URL properties from the local host. URLs for the file protocol begin with file:/ / /. The file protocol is useful for testing code on a computer that is not connected to a network.

In addition, the URL Access Manager provides firewall support, with support for HTTP proxy servers and for SOCKS gateways.

© Apple Computer, Inc. 4/21/99

The high-level URL Access Manager functions for downloading and uploading data include options that allow you to specify the display of a progress indicator and the display of an authentication dialog box if authentication is required.

The URL Access Manager includes support for automatic decompression of compressed files. If version 4.0 of the Stuffit Engine is installed, the URL Access Manager also automatically extracts files from Stuffit archives.

The URL Access Manager defines a set of URL properties. Extensions can define additional properties and set their values.

High-level URL Manager Functions

The URL Access Manager provides two high-level functions for downloading data from HTTP, HTTPS, FTP, and file:/ / / URLs:

- `URLSimpleDownload` allows you to download data synchronously from a URL into a file, a directory, or into memory.
- `URLDownload` allows you to download data synchronously from a URL reference into a file, a directory, or into memory. Using a URL reference to identify the URL allows you to obtain and set information about the URL, such its file type and creator, an associated user name and password, and the HTTP method, header, body, and user agent.

Both functions allow you to specify an application-defined system event callback routine that the URL Access Manager calls in order to convey system events to your application during the download process. You can control whether an existing file is replaced, whether a progress indicator is displayed during the transfer, and whether an authentication dialog is displayed if the URL requires authentication. You can also specify that encoded files are to be decoded and expanded if the Stuffit Engine™ is installed, indicate that the URL is a directory, or specify that you want to download a directory listing instead of the contents of a file or directory.

The URL Access Manager provides two high-level functions for uploading data to an FTP URL:

- `URLSimpleUpload` allows you to upload data synchronously from a file or a directory.

© Apple Computer, Inc. 4/21/99

- `URLUpload` allows you to upload data synchronously from a file or directory to a URL reference. As with `URLDownload`, specifying a URL reference allows you to obtain and set information about the URL before you call `URLUpload`.

Both functions allow you to specify that the data is to be encoded in BinHex format, that a progress indicator is to be displayed, and that an authentication dialog box is to be displayed if the URL requires authentication.

Low-level URL Manager Functions

The low-level URL Access Manager function, `URLOpen`, allows you to asynchronously download data from or upload data to a URL. The `URLOpen` function gives you more control over the data transfer than provided by the synchronous URL Access Manager functions.

For download operations, if you do not specify a file in which to store the data, you repeatedly call `URLGetBuffer` which retrieves the next buffer of data from the URL Access Manager's buffers so that you can manipulate the data or write it to the destination of your choice. When you call `URLOpen` in this way, you can define an event notification routine and the events for which you want to receive notification. The URL Access Manager includes definitions for URL-specific events, such as when data becomes available for downloading, when a transfer is complete, and when a certain percentage of data has been transferred.

When you call `URLOpen`, you can specify whether existing files are to be replaced, whether to encode or decode files in BinHex format, whether to display an authentication dialog box if the URL requires authentication, and whether to display a progress indicator.

© Apple Computer, Inc. 4/21/99

© Apple Computer, Inc. 4/21/99

Using the URL Access Manager

You use the URL Access Manager to transfer data to and from a Universal Resource Locator (URL).

For download operations, the URL can be a File Transfer Protocol (FTP) URL (beginning with `ftp://`), a Hypertext Transfer Protocol (HTTP) URL (beginning with `http://`), a secure HTTP URL (beginning with `https://`), or a URL that represents a local file (beginning with `file:///`).

For upload operations, the URL must be an FTP URL.

The URL Access Manager provides high-level and low-level functions for downloading and uploading data. The high-level functions include `URLSimpleDownload` and `URLSimpleUpload` for downloading and uploading data synchronously. To use these functions, specify the URL as a character string. These functions are easy to use because they allow you to start a download or upload operation with a minimal amount of preparation or intervention.

The `URLDownload` and `URLUpload` functions also allow you to download and upload data synchronously. These functions differ from `URLSimpleDownload` and `URLSimpleUpload` in that you use a URL reference to specify the URL. Using a URL reference allows you to get and set information associated with a URL by calling the `URLGetProperty` and `URLSetProperty` functions.

The `URLOpen` function is the low-level URL Access Manager function. It allows you to download data from and upload data to a URL asynchronously. When you call `URLOpen` to download data, you must call `URLGetBuffer` to transfer data to your own buffers.

This chapter presents the following examples of using the URL Access Manager:

- "Using the HTTP Post Method to Obtain Data to Download" (page 12). The sample application shows you how to use the URL Access Manager to post information to an HTTP URL and download the URL's response.

© Apple Computer, Inc. 4/21/99

- "Downloading Data from Multiple URLs" (page 18). The sample application shows you how to call `URLDownload` to download data from a series of URLs.
- "Displaying a URL's Properties" (page 21). The sample routine shows you how to obtain and display the URL Access Manager properties of a URL.

This chapter ends with a section on using AppleScript to call the URL Access Manager and a section on creating new URL Access Manager properties for a URL.

Using the HTTP Post Method to Obtain Data to Download

The `SamplePost` application calls `URLDownload` to download information from the InterNIC's `whois` database. The file `SamplePost.h`, shown in Listing 2-1, defines the URL (`kSampleURL`) from which data is to be downloaded and the structure `urlDownInfo`, which contains all of the information needed to download data from the sample URL.

The `SamplePost.h` file also includes a declaration of the routine `DoSamplePost`, which does the actual download, and a system event callback routine, `MyURLCallbackProc`, which is a place holder for code that handles system events that occur during the download.

Listing 2-1 The SamplePost application's SamplePost.h file

```
#define kSampleURL  "http://www.internic.net/cgi-bin/itts/whois"

typedef struct urlDownInfo *URLDownInfoPtr;

typedef struct urlDownInfo {
    URLReference            urlRef;
    FSSpec *                destination;
    Handle                  destinationHandle;
    URLOpenFlags            openFlags;
    URLSystemEventProcPtr   eventProc;
    void *                  userContext;
    Boolean                 done;
    OSStatus                errorCode;
} URLDownloadInfo;
```


© Apple Computer, Inc. 4/21/99

```
static void DoSamplePost ();

pascal OSStatus MyURLCallbackProc( void*, EventRecord* );
```

The `SamplePost` application is multi-threaded, so it calls `MaxApplZone` and `MoreMasters` in its main routine, as shown in Listing 2-2.

Listing 2-2 The SamplePost application's main routine

```
#include <stdio.h>
#include <Events.h>
#include <Threads.h>
#include <Processes.h>
#include <Files.h>
#include "URLAccess.h"
#include "SamplePost.h"

int main (void)
{
    OSStatus err        = noErr;

// Call MaxAppleZone() when using the Thread Manager.
    MaxApplZone();

    for (i = 0; i < 20; i++) {
        MoreMasters();
    }
```

In Listing 2-3, the `SamplePost` application verifies that the URL Access Manager is available. If the URL Access Manager is available, `DoSamplePost`, which calls `URLDownload`, is called.

Listing 2-3 Verifying the availability of the URL Access Manager

```
// Make sure the URL Access Manager is available.
    if ( URLAccessAvailable()) {
        DoSamplePost();
    }
```


© Apple Computer, Inc. 4/21/99

```
    else {
        // Call error handling routine.
    }
}
```

In Listing 2-4, the `DoSamplePost` routine defines a `URLDownloadInfo` structure named `myRef` that is uses to store information for calling `URLDownload`. The `DoSamplePost` routine calls `NewHandle` to allocate the memory in which the downloaded information will be stored. Then `DoSamplePost` creates a URL reference and stores it in `myRef.urlRef`, which will be used as a parameter when `DoSamplePost` calls `URLDownload`.

Listing 2-4 Allocating memory and creating the URL reference

```
static void DoSamplePost ( void)
{
    OSStatus err            = noErr;
    ThreadID threadID       = 0;
    URLDownloadInfo         myRef;
    Handle downloadHandle   = nil;
    long downloadSize       = 0;

    printf( "<•>DoSamplePost() Enter\n");
    downloadHandle = NewHandle(0);
    if ( downloadHandle == nil) {
        // Call error handling routine.
    }

    //  Create a URLReference
    err = URLNewReference( kSampleURL, &myRef.urlRef );
        if ( err != noErr) {
            // Call error handling routine.
        }
```

Next, `DoSamplePost` completes the preparation of the URL reference by calling `URLSetProperty` to set the value of the URL reference's HTTP Request Method property to `POST` (a 4-byte string) and the value of the URL reference's HTTP Request Body property to `whois_nic=apple.com` (a 19-byte string), as shown in Listing 2-5.

© Apple Computer, Inc. 4/21/99

Listing 2-5 Setting URL Access HTTP properties

```
URLSetProperty(myRef.urlRef, kURLHTTPRequestMethod, "POST", 4);
URLSetProperty(myRef.urlRef, kURLHTTPRequestBody, "whois_nic=apple.com", 19);
```

Note
When you set the HTTP Request Body property, the URL Access Manager automatically adds the length of the HTTP Request Header to the request, so you do not need to set the HTTP Request Header property explicitly. ◆

Listing 2-6, `DoSamplePost` uses the remaining fields of the `myRef` structure to store values that will be used as parameters for calling `URLDownload`.

Listing 2-6 Setting the URLDownload parameters

```
myRef.destination = nil;
myRef.destinationHandle = downloadHandle;
myRef.openFlags = kURLDisplayProgressFlag;
myRef.eventProc = &MyURLCallbackProc;
myRef.userContext = "1";
myRef.errorCode = 0;
```

Specifically, `DoSamplePost` does the following in Listing 2-6:

- Sets `myRef.destination` to `nil`. When `nil` is provided as the `destination` parameter to the `URLDownload` function, the calling application indicates that the downloaded data is not going to be written to a file on disk.
- Sets `myRef.destinationHandle` to the value of `downloadHandle`, which is the location in memory at which the downloaded data is to be stored.
- Sets `myRef.OpenFlags` to `kURLDisplayProgressFlag`. When the value of the `openFlags` parameter to `URLDownload` is `kURLDisplayProgressFlag`, the `URLDownload` function displays a progress indicator during the download process.
- Sets `myRef.eventProc` to the address of the `SamplePost` application's system event callback routine. When `DoSamplePost` calls `URLDownload`, it will specify `myRef.eventProc` as the `eventProc` parameter. If a system event occurs while

© Apple Computer, Inc. 4/21/99

the progress indicator is displayed, the URL Access Manager will call the routine specified by the `eventProc` parameter and will pass to it the value of the `userContext` parameter, which is described next.

- Sets `myRef.userContext` to 1. When `DoSamplePost` calls `URLDownload`, it will specify `myRef.userContext` as the `userContext` parameter. Your application can use the user context to associate any particular call of `URLDownload` with any particular call of the system event callback routine.

With the URL reference created, its properties set, and the `URLDownload` parameters prepared, the `DoSamplePost` routine is ready to call `URLDownload`, as shown in Listing 2-7.

Listing 2-7 Calling the URLDownload function

```
err = URLDownload( myRef.urlRef,
    myRef.destination,
    myRef.destinationHandle,
    myRef.openFlags,
    myRef.eventProc,
    myRef.userContext);
myRef.errorCode = err;

if (  myRef.errorCode != noErr) {
    // Call error handling routine.
}
else {
// Successful download. Get the size of the downloaded data.
err = URLGetProperty(myRef.urlRef,
    kURLResourceSize, &downloadSize, 4);
if ( err != noErr) {
    // Call error handling routine.
}
```

In Listing 2-7, if the download is successful, the `DoSamplePost` routine calls `URLGetProperty` to obtain the size of the downloaded data using the `downloadSize` parameter, which is 4 bytes long.

In Listing 2-8, `DoSamplePost` calls `SetHandleSize` to set the size of `downloadHandle` to `downloadSize` + 1 and sets the value of the last byte of downloaded data to

© Apple Computer, Inc. 4/21/99

NIL. Now that data can be displayed by calling printf. The DoDownload routine concludes by disposing of the URL reference.

Listing 2-8 Displaying the downloaded data

```
downloadSize = GetHandleSize(downloadHandle);
SetHandleSize(downloadHandle, (downloadSize+1));
(*myRef.destinationHandle)[downloadSize] = nil;

printf( "<•>==================== Downloaded Data ==================\n" );
printf( "%s", *myRef.destinationHandle );
DisposeHandle(downloadHandle);
URLDisposeReference(myref.urlRef);
    }
}
```

Listing 2-9 shows the SamplePost application's system event callback routine.

Listing 2-9 The SamplePost application's system event callback routine

```
pascal OSStatus MyURLCallbackProc( void *userContext, EventRecord *event
)
{
    printf( "<•>System callback thread fired! Thread: %u\n", userContext
);
    return 0;
}
```

The system event callback routine receives in userContext the value of the userContext parameter that was passed when the application called URLDownload.

Your application's system event callback routine should process the event record passed by the event parameter and return 0.

© Apple Computer, Inc. 4/21/99

Note
The only restriction that the URL Access Manager imposes on the functionality of your application's system event callback routine is that it should not call `URLDisposeReference`. ◆

Downloading Data from Multiple URLs

The `Downloader` application downloads data from multiple URLs in a predetermined order, and stores the data in multiple files. The application obtains the URLs that it downloads by reading a text file in which the URLs have been stored.

The `Downloader` application's main routine sets up the application's main event loop. It calls the `getURL` routine to read obtain a URL from a file of URLs and is ready to start downloading data.

Listing 2-10 The Downloader application's main routine

```
#include <Events.h>
#include <stdio.h>
#include "URLAccess.h"
#include "string.h"
#include "Memory.h"

void main (void)
{
    OSStatus err = noErr;
    char url[255];
    int count, fileCount = 0;
    EventRecord ev;
    // Call MaxApplZone, MoreMasters.
    // Initialize graph port, fonts, menus, cursor, and dialogs.
    // Clear the screen.

    while ( url != nil ) {
        // Handle Events through each loop
        WaitNextEvent(everyEvent, &ev, 0, nil);
```


© Apple Computer, Inc. 4/21/99

```
        eventHandler( nil, &ev );

        // Obtain a URL from the file of URLs.
        result = getURL(url); // (Routine not shown.)
        if ( result == eofErr ) { // Handle error condition. }

        // Call Download routine.
        result = DoDownload( url );
        if ( result != noErr) { // Handle error condition. }
    }
    printf("\n All of the URLs have been downloaded.\n");
}
```

The `DoDownload` routine shown in Listing 2-11 does the actual work of downloading data from the URL. It creates a file specification for the data that is to be downloaded and a URL reference. It sets the open flags to replace an existing file (if any) with the downloaded data and to display a progress indicator during the download. Then the `DoDownload` routine calls `URLDownload` to download the data.

Listing 2-11 The Downloader application's DoDownload routine

```
void DoDownload (void)
{
    URLReference urlRef;
    FSSpec dest, *destPtr = nil;
    destPtr = &dest;
    Handle destHandle = nil;
    int openFlags = kURLReplaceExistingFlag + kURLDisplayProgressFlag;
    Str255 newFile;

    // Create the file specification for the download.
    sprintf((char*)newFile, "File %d", fileCount);
    c2pstr((char*)newFile);
    fileCount++;
    err = FSMakeFSSpec(0, 0, newFile, &dest);

    // Create the URLReference.
    err = URLNewReference( theURL, &urlRef );
    if (err != noErr) printf("URLNewReference failed\n");
```


© Apple Computer, Inc. 4/21/99

```
    // Download the data.
    err = URLDownload( urlRef, destPtr, destHandle, openFlags,
    &eventHandler, (void*)&fileCount );

    if (err != noErr) printf("URLDownload failed\n");

    // Clean up.
    err = URLDisposeReference( urlRef );
    if (err != noErr) printf("URLDisposeReference failed\n");
    return err;
}
```

The `Downloader` application uses the `eventHandler` routine as a general purpose event handling routine. In this example, the user context (`context`) is an integer value.

Listing 2-12 The Downloader application's system event callback routine

```
pascal long eventHandler( void * userContext, EventRecord* eventPtr )
{
    EventRecord* ev;
    int what = 0;
    int context = 0;
    int* intPtr = nil;

// Convert the event pointer into an event record.
    ev = (EventRecord*)eventPtr;
    what = ev->what;
// Convert the void* to an integer.
    intPtr = (int*)userContext;
    context = *intPtr;
    if (context < 0 || context > 99)
        context = -1; // Unknown context
switch (what) {
    case 0 : // Null Event
        break;
    case mouseDown:
        printf("Handler Called: mouseDown  User Context: %d\n", context);
        // Call routine to handle event.
        break;
```


© Apple Computer, Inc. 4/21/99

```
    case updateEvt:
        printf("Handler Called: updateEvt  User Context: %d\n", context);
        // Call routine to handle event.
        break;
    case activateEvt:
        printf("Handler Called: activateEvt  User Context: %d\n",
            context);
        // Call routine to handle event.
        break;
    case keyDown:
        printf("Handler Called: keyDown  User Context: %d\n", context);
        // Call routine to handle event.
        break;
    default:
        printf("Handler Called: Default  User Context: %d\n", context);
        break;
    }
    return nil;
}
```

Displaying a URL's Properties

Given a URL reference, the `displayProperties` routine creates a `propertyList` array with one element for each of the twenty-one URL Access properties defined by Apple Computer. Then the routine gets the size of each property by calling `URLGetPropertySize`, gets the value of each property by calling `URLGetProperty`. and displays each property value.

Listing 2-13 Displaying the value of each URL property

```
void displayProperties( URLReference urlRef )
{
    OSErr err = noErr;
    int propCount = 0;
    const char* propertyList[21];
    Size propertySize = 0;
    Handle theProperty = nil;
```


© Apple Computer, Inc. 4/21/99

```
propertyList[0]  = kURLURL;
propertyList[1]  = kURLResourceSize;
propertyList[2]  = kURLLastModifiedTime;
propertyList[3]  = kURLMIMEType;
propertyList[4]  = kURLFileType;
propertyList[5]  = kURLFileCreator;
propertyList[6]  = kURLCharacterSet;
propertyList[7]  = kURLResourceName;
propertyList[8]  = kURLHost;
propertyList[9]  = kURLAuthType;
propertyList[10] = kURLUserName;
propertyList[11] = kURLPassword;
propertyList[12] = kURLStatusString;
propertyList[13] = kURLIsSecure;
propertyList[14] = kURLCertificate;
propertyList[15] = kURLTotalItems;
propertyList[16] = kURLHTTPRequestMethod;
propertyList[17] = kURLHTTPRequestHeader;
propertyList[18] = kURLHTTPRequestBody;
propertyList[19] = kURLHTTPRespHeader;
propertyList[20] = kURLHTTPUserAgent;

// Get the size of each property, allocate a handle to store the
// property's value, get the property value, and display it.
for( propCount = 0; propCount < 21; propCount++)
{
    // Get the size of the property's value.
    err = URLGetPropertySize(urlRef, propertyList[propCount], &propertySize);
    if(err != noErr)
        printf("Error %d getting property size %s. Size returned
            was: %d\n", err, propertyList[propCount], propertySize);
    else
        printf("Property size is %d: %s\n", propertySize);

    // Now get a handle for the property value.
    theProperty = NewHandleClear( propertySize + 1 );
    err = MemError();
    if(err != noErr)
        printf("Error %d getting property handle %s\n", err,
```


© Apple Computer, Inc. 4/21/99

```
                propertyList[propCount]);
        else
            printf("Got handle for %s: %s\n", propertyList[propCount]);

        // Now get the property's value.
        err = URLGetProperty(urlRef, propertyList[propCount],
                *theProperty, propertySize);
        if(err != noErr)
            printf("Error %d getting property %s\n", err, propertyList[propCount]);
        else
            printf("Property %s: %s\n", propertyList[propCount], *theProperty);

        // Clean up.
        DisposeHandle(theProperty);
        printf("\n");
    }
    return;
}
```

Using the URL Access Manager with AppleScript

You can use AppleScript to call URL Access Manager functions.

If your AppleScript application uses the URL Access Manager for operations that may take a substantial amount of time, such as transferring large amounts of data over a low-speed connection, be sure to set the timeout to large value. Setting the timeout to a large value , such as 60,000 seconds, will avoid unnecessary AppleEvent errors.

For information about the standard scripting addition commands distributed with AppleScript, see the AppleScript section of the Mac OS Help Center, or visit the following web site: http://www.apple.com/applescript/

© Apple Computer, Inc. 4/21/99

Creating New URL Access Manager Properties

In addition to the default properties described in “Property Name Constants” (page 64), extensions can create additional properties by calling `URLSetProperty` and specifying the name of a new property in the `propertyBuffer` parameter.

To prevent name collisions with properties created by other applications, the name of any property that you create should begin with a sequence of characters that reflect the name of your company, such as “MyCompany: Version 1.”

© Apple Computer, Inc. 4/21/99

CHAPTER 3

URL Access Manager Reference

This chapter describes the functions, structures, and data types that you can use to call 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 25)
- "Downloading From and Uploading to a URL Synchronously" (page 26)
- "Controlling an Asynchronous URL Upload or Download" (page 36)
- "Getting and Setting URL Properties" (page 43)
- "URL Access Manager Utility Functions" (page 46)
- "URL Access Manager Application-Defined Routines" (page 51)

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.

© Apple Computer, Inc. 4/21/99

URLGetURLAccessAvailable

Determines whether the URL Access Manager is available.

```
Boolean URLAccessAvailable ();
```

function result The `URLAccessAvailable` 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 67).

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 27) downloads data from a URL into a file or directory.
- `URLSimpleUpload` (page 29) uploads data from a file or directory to a URL.
- `URLDownload` (page 31) downloads data from a URL reference.
- `URLUpload` (page 34) uploads data to a URL reference.

© Apple Computer, Inc. 4/21/99

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 of type `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 the URL 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 file specification. 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:

© Apple Computer, Inc. 4/21/99

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

See “Open Flag Constants” (page 56) for descriptions of these constants.

`eventProc` A value of type `URLSystemEventProcPtr` that points to an application-defined system event callback routine (page 3-53) 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 dialog box, these items 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 dialog box,these items 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 67).

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.

© Apple Computer, Inc. 4/21/99

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 `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 the `url` parameter specifies a file, the file is downloadedregardless of whether `kURLDirectoryListingFlag` or `KURLIsDirectoryHintFlag` is set in the `openFlags` parameter.

When `URLSimpleDownload` downloads data from a file:/ / / URL, the data fork is downloaded but the resource fork is not downloaded.

Call `URLDownload` (page 31) if you need to set properties or `URLOpen` (page 37) 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 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 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

© Apple Computer, Inc. 4/21/99

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 "Open Flag Constants" (page 56) for descriptions of these constants.

`eventProc` A value of type `URLSystemEventProcPtr` that points to an application-defined system event callback routine (page 3-53) 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 dialog box, these items 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 dialog box, these items appear in a nonmovable 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 67).

© Apple Computer, Inc. 4/21/99

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 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 `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`.

Call `URLUpload` (page 34) if you need to set properties or `URLOpen` (page 37) if you need to control the upload process.

URLDownload

Downloads data synchronously from a URL into a file, a directory, or memory 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 54) that specifies the URL to be downloaded. Call `URLNewReference` (page 47) to create the reference.

`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.

© Apple Computer, Inc. 4/21/99

If `destination` is of type `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 the URL 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 file specification. 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 “Open Flag Constants” (page 56) for descriptions of these constants.

`eventProc` A value of type `URLSystemEventProcPtr` that points to an application-defined system event callback routine (page 3-53) 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 dialog box, the these items appear in a movable modal dialog box.

© Apple Computer, Inc. 4/21/99

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 dialog box, these items appear in a nonmovable 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 67).

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`.

If the URL reference specifies a file, the file is downloadedregardless of whether `kURLDirectoryListingFlag` or `KURLIsDirectoryHintFlag` is set in the `openFlags` parameter.

When `URLDownload` downloads data from a file:/ / / URL, the data fork is downloaded but the resource fork is not downloaded.

© Apple Computer, Inc. 4/21/99

▲ **WARNING**
Once you call `URLDownload` with `urlRef`, you cannot use the same `urlRef` to call `URLDownload` again, or to call `URLUpload` or `URLOpen`. If you need to call `URLDownload`, `URLUpload`, or `URLOpen` with a `urlRef` that has already been used for one of these calls, you need to create a new URL reference by calling `URLNewReference` (page 47).

Call `URLOpen` (page 37) if you need to control the download process.

URLUpload

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

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

`urlref` A reference of type `URLReference` (page 54) that specifies the URL to which this file or directory is to be uploaded. Call `URLNewReference` (page 47) to create the reference.

`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 “Open Flag Constants” (page 56) for descriptions of these constants.

© Apple Computer, Inc. 4/21/99

`eventProc`	A value of type `URLSystemEventProcPtr` which points to an application-defined system event callback routine (page 3-53) 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 dialog blox, these items 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 dialog box, these items 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 67).

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`.

If the URL identified by the `urlRef` parameter is a file, the file is uploaded regardless of whether `kURLDirectoryListingFlag` or `KURLIsDirectoryHintFlag` is set in the `openFlags` parameter.

© Apple Computer, Inc. 4/21/99

▲ **WARNING**

Once you call `URLUpload` with `urlRef`, you cannot use the same `urlRef` to call `URLUpload` again, or to call `URLDownlload` or `URLOpen`. If you need to call `URLUpload`, `URLDownload`, or `URLOpen` with a `urlRef` that has already been used for one of these calls, you need to create a new URL reference by calling `URLNewReference` (page 47).

Call `URLOpen` (page 37) 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 27) and `URLSimpleUpload` (page 29) and they are asynchronous, returning control to your application immediately.

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

- `URLOpen` (page 37) 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 39) 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 40) determines the amount of data that your application can retrieve from buffers.
- `URLGetBuffer` (page 41) retrieves the next buffer of data in a download operation.
- `URLReleaseBuffer` (page 42) releases the buffer obtained by a call to `URLGetBuffer`.

© Apple Computer, Inc. 4/21/99

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 54) that specifies the URL you want to open. Call `URLNewReference` (page 47) 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 a file specification for upload operations.

When `fileSpec` is a file specification, `URLOpen` automatically starts and completes the transfer of data between the URL specified by the `urlRef` parameter and the specified file. Your application will not receive data-related events, will not have access to buffer state information, and cannot call `URLGetBuffer`, `URLGetDataAvailable`, or `URLReleaseBuffer`.

If you specify `NULL` for a download operation, your application must call `URLGetBuffer` (page 41) to retrieve the data as it is downloaded.

When `fileSpec` is a file specification that specifies a file that exists and you want to replace it, specify the path to the destination file, terminate the path with a 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, you 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.

© Apple Computer, Inc. 4/21/99

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

`openFlags` A value of type `URLOpenFlags` (page 3-56) 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 "Open Flag Constants" (page 56) for descriptions of these constants.

`notifyProc` A pointer to an application-defined event notification routine as described in "Notification Callback Routine" (page 51) 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 48) 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 "Event Constants" (page 59) 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 67).

© Apple Computer, Inc. 4/21/99

DISCUSSION

The `URLOpen` function starts an asynchronous download or upload operation and returns immediately. If the `fileSpec` parameter is a valid file specification, 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 3-41) to complete the data transfer.

When `URLOpen` downloads data from a file:/ / / URL, the data fork is downloaded but the resource fork is not downloaded.

▲ **WARNING**
Once you call `URLOpen` with `urlRef`, you cannot use the same `urlRef` to call `URLOpen` again, or to call `URLDownload` or `URLUpload`. If you need to call `URLOpen`, `URLDownload`, or `URLUpload` with a `urlRef` that has already been used for one of these calls, you need to create a new URL reference by calling `URLNewReference` (page 47).

URLAbort

Terminates a data transfer.

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

`urlRef`	A reference of type `URLReference` (page 54) 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 67).

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 3-48) to `kURLAbortingState`. If you

© Apple Computer, Inc. 4/21/99

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 3-48) 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 54) that identifies the URL whose buffer size you want. Call `URLNewReference` (page 47) to create the reference.

`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 67).

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 41). 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. ▲

© Apple Computer, Inc. 4/21/99

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 44) and specify the `kURLResourceSize` property in `URLCallbackInfo` (page 55). You can then subtract the amount of data copied by `URLGetBuffer` (page 41) 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 54) 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 67).

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 transferred data.

Each buffer returned by `URLGetBuffer` is provided by the URL Access Manager, so your application should call `URLReleaseBuffer` (page 3-42) as soon as possible to prevent the URL Access Manager from running out of buffers.

© Apple Computer, Inc. 4/21/99

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. You should release the buffer as soon as possible by calling `URLReleaseBuffer` (page 3-42). ▲

URLReleaseBuffer

Releases a buffer that belongs to the URL Access Manager.

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

`urlRef`	A reference of type `URLReference` (page 54) 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 67).

DISCUSSION

The `URLReleaseBuffer` function releases a buffer obtained by calling `URLGetBuffer` (page 41). To prevent the URL Access Manager from running out of buffers, you should call `URLReleaseBuffer` as soon as possible after calling `URLGetBuffer` (page 41).

© Apple Computer, Inc. 4/21/99

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 43) gets the size of a property that can be retrieved by `URLGetProperty`.
- `URLGetProperty` (page 44) retrieves a property associated with a URL.
- `URLSetProperty` (page 45) 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 64). For a list of HTTP-specific properties that are defined and reserved by Apple Computer, Inc., see "HTTP and HTTPS Properties" (page 65).

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 67).

© Apple Computer, Inc. 4/21/99

DISCUSSION

The `URLGetPropertySize` function obtains the size of a property in bytes. You should call `URLGetPropertySize` before calling `URLGetProperty` (page 3-44) 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 67).

© Apple Computer, Inc. 4/21/99

DISCUSSION

Before calling `URLGetProperty`, call `URLGetPropertySize` (page 43) 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 45).

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 "Property Name Constants" (page 64) 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 67).

© Apple Computer, Inc. 4/21/99

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 47) creates a reference to a URL.
- `URLDisposeReference` (page 47) disposes of memory used by a URL reference.
- `URLGetCurrentState` (page 48) returns the state of the specified URL.
- `URLGetError` (page 49) retrieves the error that caused a download or upload operation to fail.
- `URLIdle` (page 50) gives the URL Access Manager time to refill its buffers.
- `URLGetFileInfo` (page 50) returns a Macintosh file type and creator as specified by the Internet configuration mapping table.

© Apple Computer, Inc. 4/21/99

URLNewReference

Creates a URL reference.

```
SStatus 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 reference of type `URLReference` (page 54). 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 67).

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 47).

URLDisposeReference

Releases memory used by a URL reference.

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

`urlRef`	A reference of type `URLReference` (page 54) that identifies the URL reference you want to dispose of.
function result	A result code. For a list of possible result codes, see "Result Codes" (page 67).

© Apple Computer, Inc. 4/21/99

DISCUSSION

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

▲ **WARNING**
Do not call `URLDisposeReference` from your system event callback routine or from your notification callback routine. Doing so may cause your application to stop working.

SEE ALSO

The function `URLNewReference` (page 47).

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 67).

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:

© Apple Computer, Inc. 4/21/99

```
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 37):

```
kURLDataAvailableState,
kURLTransactionCompleteState
```

For a description of these constants, see "State Constants" (page 57).

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 67).

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

© Apple Computer, Inc. 4/21/99

protocol-specific error code, or one of the error codes listed in “Result Codes” (page 67).

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 67).

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 `URLOpen` (page 37) and `URLGetBuffer` (page 41) 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 parameter contains the file’s type code.

`creator` A pointer to a value of type `OSType`. On return, this parameter contains the file’s creator code.

© Apple Computer, Inc. 4/21/99

function result A result code. For a list of possible result codes, see "Result Codes" (page 67).

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 37).
- A system event callback routine that receives system events when you transfer data synchronously by calling `URLSimpleDownload` (page 27), `URLSimpleUpload` (page 29), `URLDownload` (page 31), or `URLUpload` (page 34).

Notification Callback Routine

When you call `URLOpen` (page 37), 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);
```


© Apple Computer, Inc. 4/21/99

`userContext`	An application-defined value that your application previously passed as a parameter when it called `URLOpen` (page 37). 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 file specification. See the discussion section for details.
`callbackInfo`	A pointer to a structure of type `URLCallbackInfo` (page 55). 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 37) function to specify the events for which you want to receive notification.

If you call `URLOpen` with a `fileSpec` parameter that is a valid file specification, 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
```


© Apple Computer, Inc. 4/21/99

See “Event Constants” (page 59) for a description of each constant.

▲ **WARNING**
Do not call `URLDisposeReference` from your notification callback routine. Doing so may cause your application to stop working. Other than this restriction, your application can make any call. For example, your notification callback routine can update its human interface, allocate and deallocate memory, or call `NewThread`..

System Event Callback Routine

If your application calls `URLSimpleDownload` (page 27), `URLSimpleUpload` (page 29), `URLDownload` (page 31), or `URLUpload` (page 34), 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,
                    EventRecord *event);
```

`userContext` An untyped pointer to arbitrary data that your application previously passed to `URLSimpleDownload` (page 27) or `URLSimpleUpload` (page 29).

`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 27), `URLSimpleUpload` (page 29), `URLDownload` (page 31), or `URLUpload`

© Apple Computer, Inc. 4/21/99

(page 34), 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.

▲ **WARNING**
Do not call `URLDisposeReference` from your system event callback routine. Doing so may cause your application to stop working.

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 54) is a reference to a Universal Resource Locator (URL).
- The structure `URLCallbackInfo` (page 55) returns information required by your notification callback routine (page 3-51).

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 `URLNewReference` (page 47) to create a URL reference. You call `URLDisposeReference` (page 47) to dispose of the reference when you no longer need it.

© Apple Computer, Inc. 4/21/99

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 “Property Name Constants” (page 64) 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 Access Manager Constants

The following sections describe the URL Access Manager constants:

© Apple Computer, Inc. 4/21/99

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 file specification 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.

© Apple Computer, Inc. 4/21/99

`kURLExpandFileFlag`
: If the bit accessed by this flag is set, files in BinHex format are decoded. If the Stuffit Engine is installed in the System Folder, it is used to expand the file in any way supported by Stuffit.

`kURLDisplayProgressFlag`
: If the bit accessed by this flag is set, `URLSimpleDownload` (page 27), `URLSimpleUpload` (page 29), `URLDownload` (page 31), and `URLUpload` (page 34) 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.

State Constants

URL state constants are returned by `URLGetCurrentState` (page 48). The constants are defined in the `URLState` enumeration.

© Apple Computer, Inc. 4/21/99

```
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 37) has not yet been called.

`kURLInitiatingState`
The function `URLOpen` (page 37) 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 data is available

© Apple Computer, Inc. 4/21/99

in the buffers. The stream initially enters this state 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 aborting. The stream enters this state from the `kURLErrorOccurredState` state or as a result of calling `URLAbort` (page 39) when 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.

Event Constants

URL event constants are passed to your notification callback routine (page 3-51) 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,
```


© Apple Computer, Inc. 4/21/99

```
    kURLUploadingEvent              = kURLUploadingState,
    kURLSystemEvent                 = 29,
    kURLPercentEvent                = 30,
    kURLPeriodicEvent               = 31,
    kURLPropertyChangedEvent        = 32,
    };
typedef UInt32 URLEvent;
```

Constant Descriptions

`kURLInitiatedEvent`
: The function `URLOpen` (page 37) has been called but 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`
: A download operation is in progress.

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

`kURLCompletedEvent`
: All operations associated with calling `URLOpen` (page 37) have been completed. This event indicates the successful completion of a download or upload operation or the completion of cleanup work after aborting a download or upload operation.

`kURLErrorOccurredEvent`
: An error occurred.

`kURLDataAvailableEvent`
: Data is available in buffers.

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

`kURLUploadingEvent`
: An 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.

© Apple Computer, Inc. 4/21/99

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

`kURLPropertyChangedEvent`
A property, such as a filename or a user name, has become known or changed. For information about the properties that may cause this event to occur, see "Property Name Constants" (page 64).

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 in the `eventRegister` parameter when you call `URLOpen` (page 37). The `URLEventMask` enumerator defines these mask constants.

```
enum
{
    kURLInitiatedEventMask      = 1 << (kURLInitiatedEvent - 1),
    kURLResourceFoundEventMask  = 1 << (kURLResourceFoundEvent - 1),
    kURLDownloadingMask         = 1 << (kURLDownloadingEvent - 1),
    kURLUploadingMask           = 1 << (kURLUploadingEvent - l),
    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
```


© Apple Computer, Inc. 4/21/99

```
                                        + 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 `URLOpen` (page 37) 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 a URL reference has been accessed and is valid.

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

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

`kURLAbortInitiatedMask`
: Set the bit specified by this mask when you want to be notified that a 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 a call to `URLOpen` (page 37) 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 `URLOpen` (page 37), your notification callback routine is not called.

© Apple Computer, Inc. 4/21/99

`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 a file specification when you call the `URLOpen` (page 37), 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 `Property Name Constants` (page 64).

`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 `URLOpen` (page 37), 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.

© Apple Computer, Inc. 4/21/99

`kURLAllEventsMask` The bit accessed by this mask indicates that an event of any kind occurs. If you include a file specification when you call `URLOpen` (page 37), your notification callback function will not be called for the `kURLDataAvailableEvent` event and `kURLTransactionCompleteEvent` event.

Property Name Constants

These constants define properties that are used by the functions `URLGetProperty` (page 44), `URLGetPropertySize` (page 43), and `URLSetProperty` (page 45). Before you call `URLGetProperty` (page 44) to get the value of a property, you must call `URLGetPropertySize` (page 43) 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

The universal properties are defined by the following constants. You can call `URLSetProperty` (page 45) to set the value of some of these properties. The properties that can be set are indicated in the description that follows.

```
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”;
```


© Apple Computer, Inc. 4/21/99

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 a call to `URLOpen` (page 37). If a call to `URLOpen` (page 37) does not specify a file, `kURLFileType` returns a file type compatible with the MIME type.
`kURLFileCreator`	The file creator as specified in a call to `URLOpen` (page 37). If a call to `URLOpen` does not specify a file, `kURLFileCreator` returns a creator that is compatible with the MIME type.
`kURLCharacterSet`	The character set 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 that specifies whether 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 working with an HTTP or HTTPS URL, you can use the following constants to specify HTTP-specific properties:

© Apple Computer, Inc. 4/21/99

```
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 43) 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 43) 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 43) 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 43) 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.

© Apple Computer, Inc. 4/21/99

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 error codes and HTTP error codes, do not appear in this list.

`kURLNoErr`	0	No error
`kURLInvalidURLReferenceError`	–30770	Invalid URL reference
`kURLProgressAlreadyDisplayedError`	–30771	A dialog box with a progress indicatoris 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 yet available
`kURLUnknownPropertyError`	–30778	Invalid or undefined property
`kURLPropertyBufferTooSmallError`	–30779	The buffer is too small to receive the requested property
`kURLUnsettablePropertyError`	–30780	The property cannot be set
`kURLInvalidCallError`	–30781	Invalid call
`kURLFileEmptyError`	–30782	Resource file empty

© Apple Computer, Inc. 4/21/99

kURLExtensionFailureError	–30783	Extension fails to load
kURLInvalidConfigurationError	–30784	Invalid configuration
kURLAccessNotAvailableError	–30785	The URL Access Manager is not available

© Apple Computer, Inc. 4/21/99

Index

© Apple Computer, Inc. 4/21/99

D

E

© Apple Computer, Inc. 4/21/99

F

G

H

I, J

K

© Apple Computer, Inc. 4/21/99

L

M

N, O

P, Q

R

© Apple Computer, Inc. 4/21/99

S

T

U

V, W

Y, Z

© Apple Computer, Inc. 4/21/99