Network Services Location Manager (Legacy)

(Not Recommended)

Carbon > Networking

2006-05-23

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

Simultaneously published in the United States and Canada.

Contents

Figures

Introduction

Important: The Network Services Location Manager has been deprecated as of Mac OS v10.4. You should use Bonjour instead. For more information about Bonjour, see *Bonjour Overview*.

The Network Services Location Manager provides a protocol-independent way for applications to discover available network services with minimal network traffic. This document describes the programming interface for the Network Services Location Manager for Mac OS X 10.2.

What's New in the Network Services Location Manager

The following changes have been made in the Network Services Location Manager for Mac OS X 10.2:

- Instead of sending search, service registration, and service deregistration requests to Network Services Location plug-ins, the Network Services Location Manager now sends these requests to Open Directory.
- In previous versions of the Network Services Location Manager, the default neighborhoods was obtained by creating a value of type `NSLNeighborhood` whose `name` component was `NULL` and passing that `NSLNeighborhood` value as the `neighborhood` parameter to `NSLStartNeighborhoodLookup`. With this version of the Network Services Location Manager, passing `NULL` as the `neighborhood` parameter to `NSLStartNeighborhoodLookup` gets the default neighborhoods.
- To get top-level services only, you can now pass `NULL` as the `neighborhood` parameter to the `NSLStartServicesLookup` function.
- The `NSLGetNameFromNeighborhood` function now returns a null-terminated string containing the neighborhood's name.
- When calling `NSLMakeNewNeighborhood` to create a neighborhood, you can pass `NULL` as the protocolList parameter. Passing `NULL` means that you no longer have to create a protocol list. For compatibility with earlier versions of the Network Services Location Manager, you can still create a protocol list, but searches using the resulting neighborhood are not limited to the protocols specified in the protocol list.

Overview of the Network Services Location Manager

The Network Services Location Manager provides

- an easy way to dynamically discover traditional and non-traditional network services
- support for accepted and proposed industry standards
- a flexible, expandable architecture that can be easily leveraged by client and server applications

A wide variety of applications become easier to use when they call the Network Services Location Manager. For example,

- Instead of requiring the user to type a URL to locate a web server, a browser application that calls the Network Services Location Manager could have an “Open Location” command that polls the network for Hypertext Transfer Protocol (HTTP) servers and displays a list of HTTP universal resource locators (URLs) from which the user can select a particular URL.
- Collaboration software, such as a video-conferencing server, would register itself as an available service on the corporate Intranet. The users of client video-conferencing software could then search the Intranet for available conferences and join a particular conference without having to remember a cryptic URL or Internet Protocol (IP) address.

The Network Services Location Manager acts as an intermediary between the providers of network services and applications that want information about such services. It also registers network services that make registration requests.

You can use the Network Services Location Manager to

- add network-service search functionality to your application
- register a network service with the Network Services Location Manager so that it can be found in searches

This version the Network Services Location Manager runs only on Power PC computers on which Mac OS X is installed. Before your application calls the Network Services Location Manager, it should verify that Mac OS X is running.

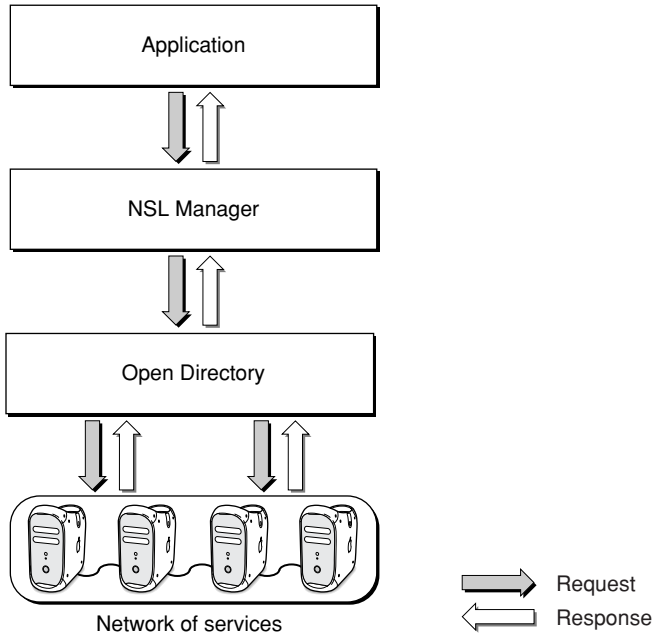
If your Mac OS X application only calls `NSLStandardGetURL`, you need only link to the Carbon library. To build a Mac OS X application that calls Network Services Location Manager functions other than `NSLStandardGetURL`, link to `/System/Library/Frameworks/CoreServices.framework`.

To build an application that calls the Network Services Location Manager and that runs on both Mac OS 9 and Mac OS X, link to the Carbon library only.

Locating Services

The Network Services Location Manager uses Open Directory to locate network services, as shown in Figure 1-1 (page 9).

Figure 1-1 Search for a network service



Applications that search for services can focus the search by specifying two values:

- a services list, a Network Services Location Manager data type that specifies one or more services that are to be searched for
- a neighborhood, a Network Services Location Manager data type that defines the boundaries for searches. Neighborhoods are established by administrators when they set up their networks for NBP and SLP. When an application calls the Network Services Location Manager's `NSLStandardGetURL` function to display a dialog box that allows users to look for services, users can supplement the neighborhoods in which searches are conducted by adding neighborhoods to a list of favorite neighborhoods. Applications that don't call `NSLStandardGetURL` call NSLStartNeighborhoodLookup to get a list of available neighborhoods.

A neighborhood is abstract information about where services may reside. In Mac OS X 10.2 and later, a neighborhood consists of one or more Open Directory "nodes." A node can be an SLP scope, a DNS domain, an AppleTalk zone, a Windows neighborhood, or a directory published by NetInfo, LDAP, or a third-party Open Directory plug-in. The Network Services Location Manager works with Open Directory to query these nodes for information about services. The Network Services Location Manager uses nodes to abstract a consistent view of the network that can be browsed in a way that is independent of the caller.

The following steps outline the conduct of a search for a service:

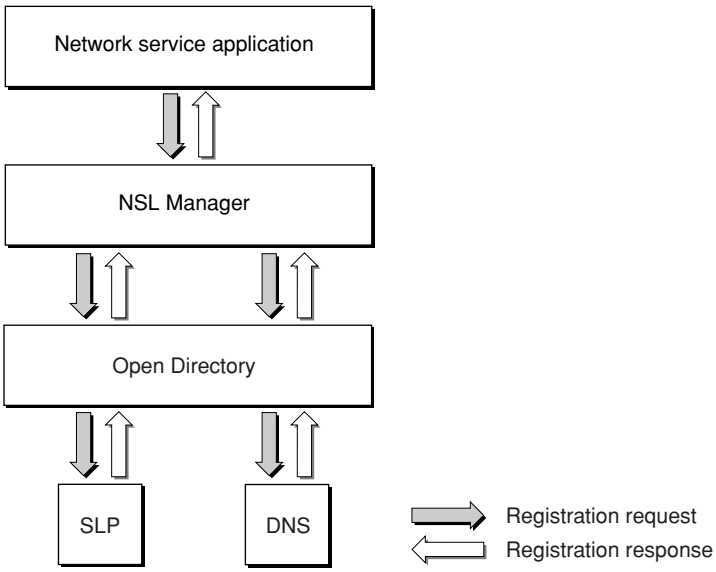
1. The application creates a search request and calls the Network Services Location Manager's `NSLStartServicesLookup` function.
2. The Network Services Location Manager receives the request and passes it to Open Directory, which passes the request to the appropriate Open Directory plug-in.
3. The Open Directory plug-in that receives the request starts to look for the requested services.

4. Providers of the requested services send their responses to the Open Directory plug-in, which makes the responses available to Open Directory.
5. Open Directory passes the responses to the Network Services Location Manager.
6. The Network Services Location Manager passes the responses to the application that called `NSLStartServiceLookup`.

Registering Services

Applications that provide services register themselves with the Network Services Location Manager as shown in Figure 1-2 (page 10) so that they can be found by applications that use their services.

Figure 1-2 Flow of a service registration



The following steps outline the flow of a service registration:

1. The application creates a value that specifies the URL to register. It may call the Network Services Location Manager utility function, `NSLHexEncodeText`, to encode portions of the URL that may contain characters (such as spaces) that are not allowed in URLs. Then the application calls the Network Services Location Manager's `NSLStandardRegisterURL` function to register the URL. In most cases, applications pass a null value as the neighborhood. Passing a null value causes the Network Services Location Manager to register the service in the appropriate local neighborhood.
2. The Network Services Location Manager receives the request and passes it to the Open Directory SLP plug-in.
3. If the Open Directory SLP plug-in can register the URL, it returns a value indicating that the service was registered successfully, which the Network Services Location Manager passes to the calling application.

Note: Open Directory may require that the application calling the Network Services Location Manager have root privileges in order to successfully register a service. SLP service registrations and service registrations in the local.arpa DNS domain are not required to have root privileges.

Tasks

The Network Services Location Manager provides high-level and low-level functions for searching for network services:

- The high-level functions are `NSLStandardRegisterURL` and `NSLStandardDeregisterURL` for registering and deregistering services and `NSLStandardGetURL`, which searches for services, displays a dialog box that lists the search results, and allows the user to choose a URL. If the user selects a URL from the search results, the `NSLStandardGetURL` function returns that URL to the calling application. Calling the high-level functions can be done without opening a session with the Network Services Location Manager or calling any other Network Services Location Manager function. For example, the NSLStandardGetURL function opens a session with the Network Services Location Manager and takes care of all of the details required to create request parameter blocks and search requests.
- Your application can call Network Services Location Manager low-level functions that open a session with the Network Services Location Manager, create request parameter blocks and search requests, and start searches.

This chapter describes the high-level Network Services Location Manager functions as well as the low-level Network Services Location Manager functions.

Registering and Deregistering Services

If your application only needs to register a network service, it can call the following Network Services Location Manager functions `NSLHexEncodeText` and without `NSLStandardRegisterURL` without having to call any other Network Services Location Manager functions.

The `NSLHexEncodeText` utility function encodes any characters that are not allowed in URLs that may occur in a URL that will be passed as a parameter to `NSLStandardRegisterURL`. For example, the space character in `http://myserver.com/`User Path is not allowed in URLs, and it needs to be encoded as `%20`. The parameters for calling `NSLHexEncodeText` are

```
OSStatus NSLHexEncodeText (
    char* rawText,
    UInt16 rawTextLen,
    char* newTextBuffer,
    UInt16* newTextBufferLen,
    Boolean* textChanged);
```

The `rawText` parameter points to the string containing the portion of a URL that is to be encoded. For example, when passed a string consisting of `User Path`, `NSLHexEncodeText` will return a string consisting of `User%20Path`. The `rawTextLen` parameter contains the length of the string pointed to by `rawText`.

When `NSLHexEncode` returns, the `newTextBuffer` parameter points to the encoded string, and the `newTextBufferLen` parameter contains the length of that string. The `textChanged` parameter indicates whether any encoding was done and is `TRUE` if the string pointed to by `newTextBuffer contains encoded characters.`

The `NSLStandardRegisterURL` function registers network services. The parameters for calling `NSLStandardRegisterURL are`

```
OSStatus NSLStandardRegisterURL (
    NSLPath urlToRegister;
    NSLNeighborhood neighborhoodToRegisterIn);
```

The `urlToRegister` parameter is a null-terminated character string containing the URL to register.

The `neighborhoodToRegisterIn` parameter specifies the neighborhood in which the service is to be registered. If the `neighborhoodToRegisterIn` parameter is `NULL`, the Network Services Location Manager determines the neighborhoods in which to register the service. For this version of the Network Services Location Manager, applications almost always pass `NULL` as the value of this parameter, thereby allowing the Network Services Location Manager to determine the appropriate local neighborhood in which to register the service.

The `NSLStandardDeregisterURL` function deregisters services that have been registered and is typically called when the system is shutting down. The parameters for calling `NSLStandardDeregisterURL` are

```
OSStatus NSLStandardDeregisterURL (
    NSLPath urlToDeregister;
    NSLNeighborhood neighborhoodToDeregisterIn);
```

The `urlToDeregister` parameter is a null-terminated string containing the URL to deregister.

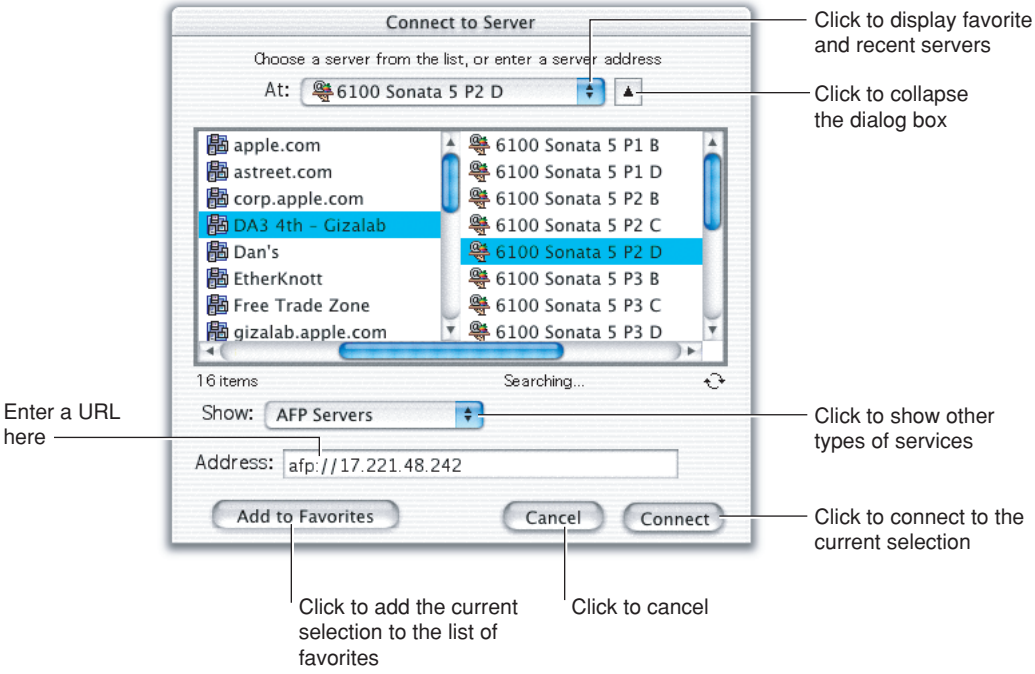
Using the NSLStandardGetURL Dialog Box

The `NSLStandardGetURL` function searches for neighborhoods and services, and displays a movable modal dialog box that lists the services and URLs that were found. The user can select a URL, and that URL is returned to the calling application.

Note: You do not need to call any other Network Services Location Manager function before calling `NSLStandardGetURL`. The `NSLStandardGetURL` function opens a session with the Network Services Location Manager, creates a service request, and closes the session with the Network Services Location Manager when it is no longer needed.

Figure 2-1 (page 15) shows the dialog box that `NSLStandardGetURL` displays.

Figure 2-1 Expanded dialog box displayed by `NSLStandardGetURL`



Clicking the up and down arrows of the At pop-up menu displays a list of favorite servers, recent servers, and neighborhoods from which users can make a selection.

The left column lists the available neighborhoods and the right column lists the servers in the selected neighborhood. The selected item in each column reflects the current selection in the At pop-up menu.

The current selection of the Show pop-up menu determines the type of service that is displayed in the At pop-up menu and the two columns. Users can use the Show pop-up menu to choose other service types (HTTP, AFP, WebDav, or FTP servers). If only one service is specified, the Show pop-up menu does not appear.

Clicking the "Add to Favorites" button adds the URL for the server that is selected in the right column to the user's Favorites folder in the Finder and causes the server to appear in the At pop-up menu. To remove a Favorite, the user selects the "Remove from Favorites" item in the At pop-up menu, which causes a dialog box to appear. The dialog box lists all of the favorites and allows the user to select a favorite and remove it by clicking the Remove button.

The user can also type a URL in the Address text field. If the Address field is not empty when the user clicks the Connect button, the content of the Address field overrides the selection in the At pop-up menu.

To connect to the selected server or to connect to the URL specified in the Address text field, the user clicks the Connect button, which causes the dialog box to be dismissed and the selected URL to be returned to the calling application.

If the user clicks the Cancel button, the dialog box is dismissed and no URL is returned to the calling application.

Applications that call `NSLStandardGetURL` can control the following features of the dialog box that `NSLStandardGetURL` displays:

- the instructional text that appears in Small System font at the top of the dialog box

- the title of the dialog box; the default title is “Connect to Server”
- the button labels whose default values are “Cancel” and “Connect”
- the list of service types that appear in the Show pop-up menu
- the service type that appears at the top of the Show pop-up menu when the dialog box is first displayed
- whether the Address field is displayed

The parameters for calling `NSLStandardGetURL` are

```
OSStatus NSLStandardGetURL (
    NSLDialogOptions* dialogOptions,
    NSLEventUPP eventProc,
    void* eventProcContextPtr,
    NSLURLFilterUPP filterProc,
    char* serviceTypeList,
    char** userSelectedURL);
```

The `dialogOptions` parameter is pointer to an `NSLDialogOptions` structure whose fields specify the appearance of the dialog box, including the title of the dialog box, button names, whether instructional text is displayed, and whether the Address field is displayed. Call the `NSLGetDefaultDialogOptions` utility function to set your `NSLDialogOptions` structure to the default values; then modify your `NSLDialogOptions` structure as desired.

The `eventProc` parameter points to an application-defined system event callback function that the Network Services Location Manager calls so that your application can handle events that may occur while the `NSLStandardGetURL` dialog box is displayed. If `eventProc` is `NULL`, your application will not receive update events.

The `eventProcContextPtr` parameter points to a value that the Network Services Location Manager passes to your system event callback function so that your application can associate any particular call of `NSLStandardGetURL` with any particular call of its system event callback function.

The `filterProc` parameter is a value of type `NSLURLFilterUPP` that points to an application-defined callback function that your application can use to filter the results that the `NSLStandardGetURL` dialog box displays.

The `serviceTypeList` is a null-terminated string of tuples that specifies the services that are to be searched for. You can use the `serviceTypeList` parameter to specify custom icons that are displayed for each service that is found. If you do not specify custom icons, default icons are displayed.

The `userSelectedURL` parameter points to a address in memory. When `NSLStandardGetURL` returns, `userSelectedURL` points to the URL of the service that was selected when the user clicked the button that by default is labeled “Connect” or points to the URL specified in the Address field. If the user clicks the button that by default is labeled “Cancel,” the value of the `userSelectedURL` parameter is a null pointer when `NSLStandardGetURL` returns. Be sure to deallocate the memory allocated for the URL by calling NSLFreeURL when you no longer need the URL.

If a error occurs, the Network Services Location Manager calls `NSLErrorToString` to get a description of the error and a solution string and then displays a dialog box containing the error description and solution string.

Using a Network Services Location Manager Session

Call `NSLOpenNavigationAPI` to open a session with the Network Services Location Manager. The `NSLOpenNavigationAPI` function returns a client reference that you use to prepare a search request.

Preparing for a Search

Call `NSLMakeNewServicesList` to prepare a comma-delimited list of services that you want to search for, such as `http`, `ftp`. If you prepare a services list and later want to add another service to it, call `NSLAddServiceToServicesList`.

Call `NSLPrepareRequest` to create a search request. The following must be provided as parameters to `NSLPrepareRequest`:

- the client reference you received when you opened the Network Services Location Manager session
- a pointer to your application's notification routine, which will be called when search results have been collected
- a pointer to an `NSLClientAsyncInfo` structure, which has fields you can use to change the conduct of searches that use the resulting search request

On output, `NSLPrepareRequest` provides a pointer to an `NSLClientAsyncInfo` structure. You can use the following fields in the `NSLClientAsyncInfo` structure to control the way in which searches are conducted:

- `maxSearchTime`, the maximum time in ticks that is to be spent on the search; zero means that the search time is not limited
- `alertInterval`, the time in ticks at which your application's notification routine is to be called even if the search is still in progress; zero means that there is no interval
- `alertThreshold`, the number of items placed in the result buffer at which your application's notification routine is to be called even if the search is still in progress; most applications set `alertThreshold` to 1

Starting a Search

To search for neighborhoods, call `NSLStartNeighborhoodLookup`. The parameters for calling `NSLStartNeighborhoodLookup` are:

- the request reference created by calling `NSLPrepareRequest`
- `NULL` to get default neighborhoods or a neighborhood value created by calling `NSLMakeNewNeighborhood` to get the names of neighborhoods related to the name specified in the neighborhood value
- a pointer to the `NSLClientAsyncInfo` structure that was provided as a parameter to `NSLPrepareRequest`

To search for services, call `NSLStartServicesLookup`. The parameters for calling `NSLStartServicesLookup` are:

- the request reference created by calling `NSLPrepareRequest`
- `NULL` to search for top-level services or a neighborhood value created by calling `NSLMakeNewNeighborhood` to search for services in the neighborhood specified by the neighborhood value
- a request parameter block created by calling `NSLMakeServicesRequestPB`; before calling `NSLMakeNewServicesPB`, you must call `NSLMakeNewServicesList` to make a services list, which specifies the services, such as `http` or `ftp`, that you want to search for
- a pointer to the `NSLClientAsyncInfo` structure that was provided as a parameter to `NSLPrepareRequest`

Processing Search Results

If you set the `maxSearchTime`, `alertInterval`, and `alertThreshold` fields of your `NSLClientAsyncInfo` structure to 0, 0, and 1, respectively, your application's notification routine is called whenever a search result is placed in the `resultBuffer` field of the `NSLClientAsyncInfo` structure.

Certain fields in the `NSLClientAsyncInfo` structure are used when a search is in progress. The fields are:

- `resultBuffer`, which points to the buffer in which the Network Services Location Manager places search results
- `bufferLen`, which reflects the number of bytes of valid data currently stored in `resultBuffer`
- `maxBufferSize`, which contains the maximum length in bytes of `resultBuffer`
- `totalItems`, which contains the number of individual search results currently stored in `resultBuffer`
- `searchState`, which describes the state of the search. The state can be that the result buffer is full, the search is ongoing, the search is complete, or the search is stalled
- `searchResult`, which consists of an `NSLError` structure whose `theErr` field may contain a result code that, depending on the state of the search, can help you decide whether to continue the search
- `searchDataType`, whose value is `kNSLNeighborhoodLookupDataEvent` if your notification routine was called as a result of a neighborhood search or `kNSLServicesLookupDataEvent` if your notification routine was called as a result of a neighborhood search

Your notification routine can use the `searchDataType` field to determine whether it should call `NSLGetNextNeighbor` or `NSLGetNextUrl` to process the result buffer.

If there are more search results to get, the search is continued when your notification routine returns. Depending on the contents of the result buffer, the value of the `searchState` and `searchResult` fields, you may decide to cancel the search by calling `NSLCancelRequest`. Note that you should not cancel the request from your notification routine.

When you no longer need a request reference, free the memory associated with it by calling `NSLDeleteRequest`, which also deallocates the `NSLClientAsyncInfo` structure associated with the deleted request reference. You'll also want to deallocate memory for any neighborhood values, services lists, and parameter blocks. The utility functions for deallocating memory are described in the next section, "Using the Network Services Location Manager Utility Functions." (page 19)

Using the Network Services Location Manager Utility Functions

In addition to `NSLMakeNewNeighborhood`, there are several Network Services Location Manager utility functions for working with neighborhood values. The utility functions that work with neighborhood values are

- `NSLCopyNeighborhood`, which makes a copy of a neighborhood value
- `NSLGetNextNeighborhood`, which obtains a pointer to the next neighborhood in a result buffer and the length of that neighborhood
- `NSLGetNameFromNeighborhood` gets the name from a neighborhood value
- `NSLGetNeighborhoodLength` gets the length of a neighborhood value

In addition to `NSLMakeServicesRequestPB` and `NSLMakeNewServicesList`, there are several utility functions for working with request parameter blocks and services lists. The utility functions that work with request parameter blocks are

- `NSLServiceIsInServiceList`, which determines whether a service is in a service list
- `NSLAddServiceToServicesList`, which adds a service to a services list

There are several Network Services Location Manager utility functions for working with URLs. The utility functions that work with URLs are

- `NSLGetServiceFromURL`, which gets the service portion of a URL
- `NSLGetNextUrl`, which obtains a pointer to the next neighborhood in a result buffer and the length of that neighborhood

Closing a Network Services Location Manager Session

Before closing a Network Services Location Manager session, you should free memory allocated for neighborhood values, services lists, request parameter blocks, and request references. The functions and utility functions for deallocating memory are

- `NSLFreeNeighborhood`, which disposes a neighborhood value
- `NSLDisposeServicesList`, which deallocates the memory associated with a services list
- `NSLFreeTypedDataPtr`, which deallocates the memory associated with a request parameter block
- `NSLDeleteRequest`, which deallocates the memory associated with a request reference, as well as the `NSLClientAsyncInfo` structure associated with the request reference

To close a Network Services Location Manager session, call `NSLCloseNavigationAPI`.

Introduction to Network Services Location Manager

Declared in NSL.h
NSLCore.h

Important: The information in this document is obsolete and should not be used for new development.

The Network Services Location Manager is an API that allows applications to locate network services. The high-level function, `NSLStandardGetURL`, automatically searches services, displays the search results in a dialog box, and allows the user to select a URL. The other high-level functions, `NSLStandardRegisterURL` and `NSLStandardDeregisterURL`, register and deregister services so they can be located by applications that search for them. Unlike the high-level Network Services Location Manager functions, the low-level Network Services Location Manager functions require you to open an session with the Network Services Location Manager, prepare search requests, check for search results, and, close the Network Services Location Manager session. The Network Services Location Manager also provides utility functions for manipulating Network Services Location Manager data types.

Network Services Location Manager Functions

NSLAddServiceToServicesList

Adds a service to a services list.

```
NSLError NSLAddServiceToServicesList (
NSLServicesList serviceList,
NSLServiceType serviceType);
```

Parameters

serviceList

On input, a value of type NSLServicesList (page 55) previously created by calling NSLMakeNewServicesList (page 37).

serviceType

On input, a value of type NSLServiceType (page 55) containing the name of the service, such as http or ftp that is to be added to the services list.

function result

An NSLError (page 53) structure whose theErr field contains the result code. If the value of NSLError.theErr is noErr, the service was added to the list. Other possible values are kNSLNotInitialized, kNSLBadServiceTypeErr, and kNSLNullListPtr. Call NSLErrorToString (page 27) to get a problem and a solution string.

Discussion

This function adds the name of the specified service to a services list. If you need to add more than one service to the services list, call this function once for each service that you want to add.

Special Considerations

You must create serviceList by calling NSLMakeNewServicesList (page 37) before calling this function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

NSLCore.h

NSLCancelRequest

Cancels an ongoing search.

```
NSLError NSLCancelRequest (NSLRequestRef ref);
```

Parameters

ref

On input, a value of type `NSLRequestRef` (page 55), obtained by previously calling `NSLPrepareRequest` (page 38), for the search that is to be canceled.

function result

An `NSLError` (page 53) structure whose `theErr` field contains the result code. A value of `noErr` indicates that the request was canceled successfully. Other possible values are `kNSLNotInitialized` and `kNSLBadReferenceErr`. Call `NSLErrorToString` (page 27) to get a problem and a solution string.

Discussion

This function cancels an ongoing search. Any outstanding I/O is also canceled.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`NSLCore.h`

NSLCloseNavigationAPI

Closes a session with the Network Services Location Manager.

```
void NSLCloseNavigationAPI (NSLClientRef theClient);
```

Parameters

theClient

On input, a value of type `NSLClientRef` (page 54), obtained by previously calling `NSLOpenNavigationAPI` (page 38), that identifies the session that is to be closed.

Discussion

This function closes the Network Services Location Manager session identified by `theClient`. If your application calls this function while a search is in progress, any data that would have been returned is lost.

Special Considerations

Your application is responsible for reclaiming memory that it allocates for services lists, parameter blocks, and search request references. Your application should reclaim memory by calling `NSLDisposeServicesList` (page 26), `NSLDeleteRequest` (page 26), and `NSLFreeTypedDataPtr` (page 28), respectively.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`NSLCore.h`

NSLContinueLookup

Continues a search.

```
NSLError NSLContinueLookup (
NSLClientAsyncInfo *asyncInfo);
```

Parameters

asyncInfo

On input, a pointer to the NSLClientAsyncInfo (page 51) structure for this search.

function result

An NSLError (page 53) structure whose theErr field contains the result code. If the value of NSLError.theErr is noErr, NSLContinueLookip returned successfully. Possible errors include kNSLNotInitialized, kNSLNoContextAvailable, kNSLBadClientInfoPtr, kNSLCannotContinueLookup, and kNSLBufferTooSmallForData. Call NSLErrorToString (page 27) to get a problem and a solution string.

Discussion

This function is deprecated in Mac OS X and returns without performing any work. In Mac OS X, asynchronous searches continue automatically when the application's notification routine returns. For compatibility with earlier versions of the Network Services Location Manager, applications can continue to call this function, but doing so is not necessary.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

NSLCore.h

NSLCopyNeighborhood

Copies a neighborhood.

```
NSLNeighborhood NSLCopyNeighborhood (
NSLNeighborhood neighborhood);
```

Parameters

neighborhood

On input, the neighborhood that is to be copied.

result

An NSLNeighborhood (page 54) value. If this function can't copy the neighborhood specified by the neighborhood parameter, it returns NULL. This might happen, for example, if there is not enough memory.

Discussion

This utility function creates a copy of the specified neighborhood. A neighborhood is guaranteed to exist only during the execution of an application's notification routine, so you may want to call this utility function from your application's notification routine in order to keep a copy of the neighborhood.

When you have no further use for an NSLNeighborhood value, you should reclaim the memory allocated to it by calling NSLFreeNeighborhood (page 28).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
NSLCore.h

NSLDeleteRequest

Deletes a request reference.

```
NSLError NSLDeleteRequest (
NSLRequestRef ref);
```

Parameters

ref

On input, the value of type NSLRequestRef (page 55), obtained by previously calling NSLPrepareRequest (page 38), that is to be deleted.

function result

An NSLError (page 53) structure whose theErr field contains the result code. If the value of the theErr field is noErr, the request reference was deleted. Other possible values are kNSLNotInitialized and kNSLBadReferenceErr. Call NSLErrorToString (page 27) to get a problem and a solution string.

Discussion
This function deletes the specified request reference and deallocates memory associated with it, including the NSLClientAsyncInfo (page 51) structure associated with the request reference. If a search using the specified request reference is in progress when this function is called, the search is canceled and any outstanding I/O is lost.

This function does not deallocate memory associated with services lists or request parameter blocks. To deallocate memory for services lists, call NSLDisposeServicesList (page 26); to deallocate memory for parameter blocks, call NSLFreeTypedDataPtr (page 28).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
NSLCore.h

NSLDisposeServicesList

Disposes of a services list.

```
void NSLDisposeServicesList (
NSLServicesList theList);
```

Parameters

theList

On input, a value of type NSLServicesList (page 55), previously created by calling NSLMakeNewServicesList (page 37), that is to be disposed of.

Discussion

This utility function reclaims memory by disposing of a services list. Once you've incorporated the information in a services list into a request parameter block by passing the services list to NSLMakeServicesRequestPB (page 37), you can dispose of the services list.

Calling NSLCloseNavigationAPI (page 24) does not reclaim memory allocated for services lists, so your application should dispose of services lists before it closes a Network Services Location Manager session.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

NSLCore.h

NSLErrorToString

Obtains information about an error.

```
OSStatus NSLErrorToString (
NSLError theErr,
char * errorString,
char * solutionString);
```

Parameters

theErr

On input, an NSLError (page 53) structure whose theErr field contains a Network Services Location Manager result code.

errorString

On input, a pointer to the buffer in which this function is to place a null-terminated string containing a description of the problem that caused the error. The length of errorString should be 256 bytes.

solutionString

On input, a pointer to the buffer in which this function is to place a null-terminated string containing a possible solution to the problem. The length of solutionString should be 256 bytes.

function result

A value of noErr indicates that this function returned successfully. If the value of the theContext field of the NSLError structure that was passed to this function is zero and the theErr field contains a result code that is not within the range of Network Services Location Manager result codes, this function returns kNSLBadReferenceErr.

Discussion

This function obtains the error and solution strings for the result code stored in the theErr field of an NSLError (page 53) structure.Your application can use the strings to display an appropriate error message.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`NSLCore.h`

NSLFreeNeighborhood

Disposes of an `NSLNeighborhood` value.

```
NSLNeighborhood NSLFreeNeighborhood (NSLNeighborhood neighborhood);
```

Parameters

`neighborhood`
On input, the `NSLNeighborhood` (page 54) value, obtained by previously calling `NSLMakeNewNeighborhood` (page 36), that is to be disposed of.

`function result`
An `NSLNeighborhood` whose value is `NULL`.

Discussion
This utility function disposes of an `NSLNeighborhood` value and reclaims the memory that was allocated to it.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`NSLCore.h`

NSLFreeTypedDataPtr

Frees memory allocated for a request parameter block.

```
NSLTypedDataPtr NSLFreeTypedDataPtr (
NSLTypedDataPtr nslTypeData);
```

Parameters

`nslTypeData`
On input, a value of type `NSLTypedDataPtr` (page 56), obtained by previously calling `NSLMakeServicesRequestPB` (page 37).

`function result`
A value of type `NSLTypedDataPtr` whose value is `NULL`.

Discussion
This utility function frees memory that was allocated when your application previously called `NSLMakeServicesRequestPB`. Your application should free the memory allocated for a request parameter block when it has no further use for the parameter block.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
NSLCore.h

NSLFreeURL

Frees memory allocated for a URL.

```
(char *) NSLFreeURL (
char * URL);
```

Parameters

URL
On input, a pointer to a character string obtained by previously calling NSLStandardGetURL (page 41).

result
NULL.

Discussion
This utility function frees the memory that was allocated for URL when NSLStandardGetURL was called. Your application should call this utility function when it has no further use for the URL.

Availability
Available in Mac OS X v10.0 and later.

Declared In
NSL.h

NSLGetDefaultDialogOptions

Assigns the default values to an NSLDialogOptions structure.

```
OSStatus NSLGetDefaultDialogOptions (
NSLDialogOptions * dialogOptions);
```

Parameters

dialogOptions
On input, a pointer to an NSLDialogOptions (page 52) structure. On output, the fields of the structure are filled in with the default values.

function result
A value of noErr indicates that the fields were filled in with the default values. See "Network Services Location Manager Result Codes" (page 61) for other possible result codes.

Discussion
This utility function assigns default values to the NSLDialogOptions structure pointed to by the dialogOptions parameter. Assigning default values prepares the structure for use when calling NSLStandardGetURL (page 41). After the default values are assigned, you can modify the structure as desired. For the default values, see NSLDialogOptions (page 52).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSL.h`

NSLGetNameFromNeighborhood

Locates the name in a neighborhood.

```
void NSLGetNameFromNeighborhood (
NSLNeighborhood neighborhood,
char ** name,
long * length);
```

Parameters

`neighborhood`

On input, a value of type `NSLNeighborhood` (page 54) containing the name that is to be located.

`name`

On input, the address of a pointer. On output, `name` contains the address of a pointer to the name in the `neighborhood` parameter.

`length`

On input, a pointer to a location in memory. On output, `length` points to the length in bytes of the name in the `neighborhood` parameter.

Discussion
This utility function locates the null-terminated name in the neighborhood specified by the `neighborhood` parameter so that your application can, for example, display the name. Use the value pointed to by `length` to determine the length of the name.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`NSLCore.h`

NSLGetNeighborhoodLength

Gets the length of a neighborhood.

```
long NSLGetNeighborhoodLength (
NSLNeighborhood neighborhood);
```

Parameters

`neighborhood`

On input, the value of type `NSLNeighborhood` (page 54) whose length is to be obtained.

`function result`

The length in bytes of the specified neighborhood.

Discussion
This utility function gets the length of the specified neighborhood, which you would need in order to make a copy of a neighborhood. Instead of calling this utility function, call `NSLCopyNeighborhood` (page 25) to make a copy of a neighborhood.

Special Considerations

This utility function may be deprecated in the future.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`NSLCore.h`

NSLGetNextNeighborhood

Gets the position of the next neighborhood in a result buffer.

```
Boolean NSLGetNextNeighborhood (
NSLClientAsyncInfoPtr infoPtr,
NSLNeighborhood * neighborhood,
long * neighborhoodLength);
```

Parameters

infoPtr

On input, a pointer to an `NSLClientAsyncInfo` (page 51) structure whose resultBuffer field may contain another neighborhood.

neighborhood

On input, a pointer to a value of type `NSLNeighborhood` (page 54). On output, `neighborhood` points to the next neighborhood in the `resultBuffer` field of the `NSLClientAsyncInfo` structure pointed to by `infoPtr`.

neighborhoodLength

On output, the length of the neighborhood pointed to by `neighborhood`.

function result

A Boolean value. A value of `TRUE` indicates that `neighborhood` points to the next neighborhood in the `resultBuffer` field pointed to by `infoPtr`. A value of `FALSE` indicates that there are no more neighborhoods in the `resultBuffer field`.

Discussion
This utility function provides the starting position and the length of the next neighborhood in a result buffer. Typically, the neighborhoods were placed in the result buffer by a previous call to `NSLStartNeighborhoodLookup` (page 44).

If you want to get a copy of the neighborhood, call `NSLCopyNeighborhood` (page 25). If you want to get the name from the neighborhood, call `NSLGetNameFromNeighborhood` (page 30).

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
NSLCore.h

NSLGetNextURL

Gets the position of the next URL in a result buffer.

```
Boolean NSLGetNextUrl (
NSLClientAsyncInfoPtr infoPtr,
char ** urlPtr,
long * urlLength);
```

Parameters

infoPtr

On input, a pointer to an NSLClientAsyncInfo (page 51) structure whose resultBuffer field may contain another URL.

urlPtr

On output, if another URL was found in the resultBuffer field, a pointer to the beginning of the URL.

urlLength

On output, the length of the URL pointed to by urlPtr.

function result

A Boolean value. A value of TRUE indicates that urlPtr points to the next URL in the resultBuffer field pointed to by infoPtr. A value of FALSE indicates that there are no more URLs in the resultBuffer field.

Discussion
This utility function obtains the starting position and the length of the next URL in a result buffer. Typically, the URLs were placed in the result buffer by a previous call to NSLStartServicesLookup (page 45).

NSLGetServiceFromURL

Gets the service portion of a URL.

```
OSStatus NSLGetServiceFromURL (
char* theURL,
char** svcString,
UInt16* svcLen);
```

Parameters

theURL

On input, a pointer to a null-terminated string containing a URL.

svcString

On input, a pointer to the address of a string in memory. On output, svcString points to the service portion of the URL specified by theURL.

svcLen

On input, a pointer to an unsigned 16-bit value. On output, svcLen points to the length in bytes of svcString.

function result

A value of `noErr` indicates that `svcString` points to the service portion of the URL specified by `theURL`.

Discussion
This utility function gets the service portion of a URL. For example, if the URL is `http://www.apple.com`, the service portion of that URL is `http`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`NSLCore.h`

NSLHexDecodeText

Decodes the encoded portion of a URL.

```
OSStatus NSLHexDecodeText (
char* encodedText,
UInt16 encodedTextLen,
char* decodedTextBuffer,
UInt16* decodedTextBufferLen,
Boolean* textChanged);
```

Parameters

encodedText

On input, a pointer to a buffer containing the portion of a URL that has been encoded.

encodedTextLen

On input, a value of type `UInt16` that specifies the length of the buffer pointed to by encodedText.

decodedTextBuffer

On input, a pointer to a buffer in which the decoded text is to be stored. On output, the buffer contains the decoded text.

decodedTextBufferLen

On input, a pointer to a value of type `UInt16` containing the maximum length of `decodedTextBuffer`. On output, `decodedTextBufferLen` points to the length of the decoded text pointed to by `decodedTextBuffer`.

textChanged

On input, a pointer to a Boolean value. On output, `textChanged` points to a value that is `TRUE` if decoding occurred or that is `FALSE` if no decoding was required.

function result

A value of `noErr` indicates that this utility function returned successfully. See "Network Services Location Manager Result Codes" (page 61) for other possible result codes.

Discussion
This utility function decodes the portion of a URL that was previously encoded by `NSLHexEncodeText` (page 34). If this utility function returns `noErr`, the buffer pointed to by `decodedTextBuffer` contains the decoded copy of the buffer pointed to by `encodedTextBuffer`, or it contains an exact copy of the buffer pointed to by `encodedTextBuffer`. Use the `textChanged` parameter to determine whether any decoding actually took place.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`NSLCore.h`

NSLHexEncodeText

Encodes a portion of a URL.

```
OSStatus NSLHexEncodeText (
char* rawText,
UInt16 rawTextLen,
char* newTextBuffer,
UInt16* newTextBufferLen,
Boolean* textChanged);
```

Parameters

rawText
On input, a pointer to a character array containing the portion of a URL that is to be encoded. For example, if the URL is `afp://17.221.40.66?NAME=Kevs G3`, the portion of the URL that contains a character that is not allowed is `Kevs G3`. In this example, the space character is not allowed.

rawTextLen
On input, a value of type `UInt16` containing the length in bytes of the `rawText` parameter.

newTextBuffer
On input, a pointer to a buffer. On output, the buffer contains the encoded text.

newTextBufferLen
On input, a pointer to a value of type *UInt16*. On output, `newTextBufferLen` points to the length of the encoded text in the buffer pointed to by `newTextBuffer`.

textChanged
On input, a pointer to a Boolean value. On output, `textChanged` points to a value that is `TRUE` if encoding occurred or that is `FALSE` if no encoding was required.

function result
A value of `noErr` indicates that this utility function returned successfully. See "Network Services Location Manager Result Codes" (page 61) for other possible result codes.

Discussion
This utility function uses the US ASCII character set to encode characters that are not allowed in URLs. Hexadecimal values from 01 to 1F, from 80 to FF, and 7F are not allowed. In addition, the following characters are not allowed:

< > " # { } | \ ^ ~ [ ]

This utility function also encodes the following characters that are reserved for URL syntax:

; / ? : @ = % &

If this utility function returns `noErr`, the buffer pointed to by `newTextBuffer` contains the encoded copy of the buffer pointed to by `rawText`, or it contains an exact copy of the buffer pointed to by `rawText`. Use the `textChanged` parameter to determine whether any encoding actually took place.

Call `NSLHexDecodeText` (page 33) to decode a string that has been encoded.

Special Considerations

The `NSLStandardRegisterURL` (page 43) function returns an error if a URL you try to register contains characters that need to be encoded.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`NSLCore.h`

NSLLibraryPresent

Determines whether the Network Services Location Manager is present.

```
Boolean NSLLibraryPresent (void);
```

Parameters

function result
`TRUE` if the Network Services Location Manager is present.

Discussion
This utility function returns `TRUE` when the Network Services Location Manager is available. The Network Services Location Manager is always present in Mac OS X, so for applications that only run on Mac OS X do not need to call this function.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.3.

Declared In
`NSLCore.h`

NSLLibraryVersion

Gets the Network Services Location Manager's version.

```
UInt32 NSLLibraryVersion (void);
```

Parameters

function result
The Network Services Location Manager's version number in hexadecimal format, where the first two bytes represent the version number, the second two bytes represent the revision number, and the third two bytes represent the subrevision number.

Discussion
This utility function gets the version of the Network Services Location Manager installed on the system.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
NSLCore.h

NSLMakeNewNeighborhood

Creates a neighborhood for use in searches.

```
NSLNeighborhood NSLMakeNewNeighborhood (
char * name,
char * protocolList);
```

Parameters

name

On input, NULL to create a neighborhood that can be used to obtain a list of default neighborhoods when calling NSLStartNeighborhoodLookup (page 44), or a pointer to a null-terminated string containing the name of a neighborhood in which searches are to be conducted.

protocolList

On input, NULL, which creates a neighborhood that will cause all available protocols to participate in any search that uses it. For compatibility with earlier versions of the Network Services Location Manager, protocolList can be a pointer to a comma-separated, null-terminated list of protocols (such as NBP,SLP) that are to participate in a search conducted with the resulting neighborhood. The constants kDSProtocolType, kNBPProtocolType, and kSLPProtocolType can be used to build a protocol list. However, searches conducted using the resulting neighborhood will not be limited to the protocols that were specified.

function result

An NSLNeighborhood (page 54) value that can be used in subsequent calls to Network Services Location Manager functions, or NULL if a neighborhood value could not be created. This might happen, for example, if there is not enough memory.

Discussion
This utility function creates a neighborhood that can be provided as a parameter to other Network Services Location Manager functions such as NSLStartNeighborhoodLookup (page 44), NSLStartServicesLookup (page 45), NSLStandardRegisterURL (page 43), and NSLStandardDeregisterURL (page 41).

When you have no further use for a neighborhood, you can reclaim the memory allocated to it by calling NSLFreeNeighborhood (page 28).

Special Considerations

Specifying protocols in the protocolList parameter is deprecated. If you specify protocols when other protocols are available, any use of the resulting neighborhood will not be limited to the protocols you specified.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
NSLCore.h

NSLMakeNewServicesList

Creates a services list.

```
NSLServicesList NSLMakeNewServicesList (
char* initialServiceList);
```

Parameters

initialServiceList

On input, a pointer to a comma-delimited, null-terminated string of service names, such as `http`, `ftp`.

function result

A value of type `NSLServicesList` (page 55). This function returns `NULL` if it can't create the services list because, for example, there is not enough memory or because the Network Services Location Manager is not initialized.

Discussion

This function creates a services list and fills it with the names of the services specified by the `initialServiceList` parameter. After creating the services list, you can add the names of additional services by calling `NSLAddServiceToServicesList` (page 23).

Special Considerations

When you have no further use for the services list, you should reclaim the memory allocated to it by calling `NSLDisposeServicesList` (page 26).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`NSLCore.h`

NSLMakeServicesRequestPB

Creates a request parameter block.

```
OSStatus NSLMakeServicesRequestPB (
NSLServicesList serviceList,
NSLTypedDataPtr * newDataPtr);
```

Parameters

serviceList

On input, a value of type `NSLServicesList` (page 55) created by previously calling `NSLMakeNewServicesList` (page 37).

newDataPtr

On input, the address of the `NSLTypedDataPtr` (page 56) at which the resulting parameter block is to be placed.

function result

A value of `noErr` indicates that the parameter block was created successfully. Possible errors include `kNSLBadDomainErr`.

Discussion
This utility function creates a parameter block formatted properly for use with subsequent calls to `NSLStartServicesLookup` (page 45).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`NSLCore.h`

NSLOpenNavigationAPI

Opens a session with the Network Services Location Manager.

```
OSStatus NSLOpenNavigationAPI (
NSLClientRef * newRef);
```

Parameters

`newRef`

On input, a pointer to a value of type `NSLClientRef` (page 54) in which the Network Services Location Manager places a client reference. Pass the client reference to `NSLPrepareRequest` (page 38) and `NSLCloseNavigationAPI` (page 24).

`function result`

A value of `noErr` indicates that the session was opened successfully. If `NSLOpenNavigationAPI` returns any of the following error codes, your application should not call any other Network Services Location Manager functions: `kNSLNotInitialized`, `kNSLInsufficientSysVer`, `kNSLInsufficientOTVer`, `kNSLPluginLoadFailed`, or `kNSL68kContextNotSupported`.

Discussion
This function opens a session with the Network Services Location Manager and returns a client reference that your application later uses to prepare search requests and to close the Network Services Location Manager session. You must call `NSLOpenNavigationAPI` before calling any other Network Services Location Manager functions.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`NSLCore.h`

NSLPrepareRequest

Creates a search request.

```
NSLError NSLPrepareRequest (
NSLClientNotifyUPP notifier,
void * contextPtr,
NSLClientRef theClient,
NSLRequestRef * ref,
char * bufPtr,
unsigned long bufLen,
NSLClientAsyncInfoPtr * infoPtr);
```

Parameters

notifier

On input, a value of type `NSLClientNotifyUPP` that points to your application's notification routine, or `NULL`. Your notification routine will be called when data is available, when the search is complete, or when an error occurs.

contextPtr

On input, an untyped pointer to arbitrary data that the Network Services Location Manager will pass to your application's notification routine when that routine is called. Your application can use `contextPtr` to associate any particular execution of your notification routine with any particular search request.

theClient

On input, a value of type `NSLClientRef` (page 54) obtained by previously calling `NSLOpenNavigationAPI` (page 38) that identifies the Network Services Location Manager session for which this request is being prepared.

ref

On input, a pointer to a value of type `NSLRequestRef` (page 55) in which the resulting search request reference is to be placed.

bufPtr

On input, a pointer to the buffer in which search results are to be placed.

bufLen

On input, the length of the buffer pointed to by `bufPtr`.

infoPtr

On output, the `NSLClientAsyncInfo` (page 51) structure pointed to by `infoPtr` contains default instructions for how the search is to be conducted. Your application can change the defaults before it starts the search.

function result

An `NSLError` (page 53) structure whose `theErr` field contains the result code. If the value of `NSLError.theErr` is `noErr`, the request was created. Other possible values include `kNSLNotInitialized`, `kNSLDuplicateSearchInProgress`, `kNSLBadClientInfoPtr`, and `kNSLErrNullPtrError` (if the value of `infoPtr` is `NULL`). Call `NSLErrorToString` (page 27) to get a problem and a solution string.

Discussion

This function creates a search request that your application later provides as a parameter when it calls `NSLStartNeighborhoodLookup` (page 44) or `NSLStartServicesLookup` (page 45). Searches are conducted asynchronously.

Your application's notification routine will be called when the result buffer contains data, the result buffer is full, when the search is complete, or when an error occurs. Your application can cause your application's notification routine to be called at a specified interval, when a specified number of items is in the result

buffer, or when a specified amount of time has elapsed by modifying the value of the `alertInterval`, `alertThreshold`, and `maxSearchTime` fields, respectively, of the `NSLClientAsyncInfo` (page 51) structure pointed to by `infoPtr`.

If this function returns a result code of `kNSLDuplicateSearchInProgress`, a search is in progress that is using a request reference that is identical to the newly created request reference. Your application can ignore this warning and use the newly created request reference, delete the newly created request reference, or cancel the search that is using the identical request reference.

Special Considerations

When your application no longer needs the request reference, it should call `NSLDeleteRequest` (page 26) to reclaim memory associated with it.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`NSLCore.h`

NSLServiceIsInServiceList

Determines whether a service is in a services list.

```
Boolean NSLServiceIsInServiceList (
NSLServicesList serviceList,
NSLServiceType svcToFind);
```

Parameters

`serviceList`
On input, a value of type `NSLServicesList` (page 55) created by previously calling `NSLMakeNewServicesList` (page 37).

`svcToFind`
On input, a value of type `NSLServiceType` (page 55) specifying the service that is to be checked.

`function result`
A value of `TRUE` indicates that the service specified by `svcToFind` is in the services list specified by `serviceList`; otherwise, the value returned is `FALSE`.

Discussion
This utility function determines whether the service specified by `svcToFind` is in the services list specified by `serviceList`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`NSLCore.h`

NSLStandardDeregisterURL

Deregisters a service.

```
OSStatus NSLStandardDeregisterURL (
NSLPath urlToDeregister,
NSLNeighborhood neighborhoodToDeregisterIn);
```

Parameters

urlToDeregister

On input, a value of type NSLPath (page 55) specifying the URL that is to be deregistered.

neighborhoodToDeregisterIn

On input, a value of type NSLNeighborhood (page 54) specifying the neighborhood in which to deregister the service, or NULL. If NULL is specified, the Network Services Location Manager determines the neighborhood from which the service is deregistered.

function result

A value of noErr indicates that the service was deregistered. A value of kNSLBadURLSyntax indicates that the URL contains characters that are not allowed, such as spaces. To encode characters that are not allowed, call NSLHexEncodeText (page 34).

Discussion

This function deregisters the service specified by urlToDeregister. An application that registers services by calling NSLStandardRegisterURL (page 43) should call this function as part of its standard shutdown procedure.

Special Considerations

You do not have to call NSLOpenNavigationAPI (page 38) before calling this function.

This function is available in version 1.1 of the Network Services Location Manager and later, and supersedes the NSLDeRegisterService function, which was provided in version 1.0.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

NSLCore.h

NSLStandardGetURL

Displays a dialog box that searches for services and allows the selection of a URL.

```
OSStatus NSLStandardGetURL (
NSLDialogOptions * dialogOptions,
NSLEventUPP eventProc,
void * eventProcContextPtr,
NSLURLFilterUPP filterProc,
char * serviceTypeList,
char ** userSelectedURL);
```

Parameters

dialogOptions

On input, a pointer to an `NSLDialogOptions` (page 52) whose fields control the appearance of the dialog box. Before calling this function, call `NSLGetDefaultDialogOptions` (page 29) to fill the fields of your `NSLDialogOptions` structure with the default values. Then customize the values of the fields of the `NSLDialogOptions` structure as desired.

eventProc

On input, a value of type `NSLEventUPP` that points to an application-defined system event callback, NSLEventProcPtr (page 49), or `NULL`. If `eventProc` is `NULL`, your application will not receive update events while the `NSLStandardGetURL` dialog box is displayed.

eventProcContextPtr

On input, an untyped pointer to arbitrary data that the Network Services Location Manager passes to the system event callback specified by `eventProc`. Your application can use `eventProcContextPtr` to associate any particular execution of your system event callback function with any particular call to this function.

filterProc

On input, a value of type `NSLURLFilterUPP` that specifies your filter callback function, NSLFilterProcPtr (page 49), or `NULL` if you do not have a filter callback function. If specified, your filter callback function will be called for each URL that is about to be displayed in the dialog box. If your filter callback function returns `TRUE`, the URL is displayed in the `NSLStandardGetURL` dialog box. If your filter callback function returns `FALSE`, the URL is not displayed.

userSelectedURL

On input, the address of a pointer to a string that will receive the URL that the user may select or enter. On output, if this function returns `noErr`, `userSelectedURL` contains the null-terminated URL the user selected. When your application no longer needs `userSelectedURL`, it should call `NSLFreeURL` (page 29) to reclaim the memory associated with it.

serviceTypeList

On input, a null-terminated string of tuples that describes the services that are to be searched for. See the Discussion section for a description of the format and for information on how to use `serviceTypeList` to control the icon that is displayed for each service type.

function result

A value of `noErr` indicates that the user selected a URL, which is stored in the `userSelectedURL` parameter. A value of `kNSLUserCanceled` indicates that the user clicked the Cancel button in the dialog box; the `userSelectedURL` parameter is empty.

Discussion

This function displays a dialog box that allows the user to select the type of service that is to be searched for and the neighborhood in which the search is to be conducted. The calling application is responsible for specifying the list of services types, and the Network Services Location Manager is responsible for displaying the neighborhoods, which it obtains by querying Open Directory.

The `serviceTypeList` parameter consists of service names and service descriptor lists in the following format: *service-name* , *service-descriptor-list* ; where *service-descriptor-list* is a comma-delimited list of services. For example, if you want to search for HTTP, HTTPS, and FTP services, the value of `serviceTypeList` would

be `Web Servers,http,https;FTP Servers,ftp` Setting `serviceTypeList` in this way would cause the Show pop-up menu in the dialog box displayed by this function to contain two items: "Web Servers" and "FTP Servers". The result of a search performed on these two items would consist of the list of HTTP and HTTPS services that were found, followed by the list of FTP services that were found.

This function displays a unique icon for each of the http, https, ftp, afp, lpr, LaserWriter, and AFPServer service descriptors and the same generic icon for any other service descriptor. You can have your own icon displayed by specifying an icon suite resource ID in the `serviceTypeList` parameter. For example, if the value of `serviceTypeList` is

```
Web Servers,http,https;Telnet Servers,telnet;NFS Servers,nfs,129
```

the Network Services Location Manager's unique icons will be displayed for HTTP and HTTPS services, the Network Services Location Manager's generic icon will be displayed for Telnet services, and the icon suite at resource ID 129 in your application's resource fork will be displayed for NFS services.

Special Considerations

Be sure to dispose of the `userSelectedURL` parameter by calling `NSLFreeURL` (page 29) when `userSelectedURL` is no longer needed.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSL.h`

NSLStandardRegisterURL

Registers the URL of a service.

```
OSStatus NSLStandardRegisterURL (
NSLPath urlToRegister,
NSLNeighborhood neighborhoodToRegisterIn);
```

Parameters

`urlToRegister`

On input, a value of type `NSLPath` (page 55) specifying the URL to register.

`neighborhoodToRegisterIn`

On input, a value of type `NSLNeighborhood` (page 54) specifying the neighborhood in which to register the service, or `NULL`. If `NULL` is specified, the Network Services Location Manager determines the neighborhood in which the URL is registered.

`function result`

A value of `noErr` indicates that the service was registered. A value of `kNSLBadURLSyntax` indicates that the URL contains characters that are not allowed, such as spaces. To encode characters that are not allowed, call `NSLHexEncodeText` (page 34).

Discussion
This function registers the specified URL with the Network Services Location Manager.

Applications that provide a network service typically call this function as part of their startup procedure. They also call `NSLStandardDeregisterURL` (page 41) as part of their shutdown procedure.

Special Considerations

You do not have to call `NSLOpenNavigationAPI` (page 38) before calling this function.

This function is available in version 1.1 and later of the Network Services Location Manager, and supersedes the `NSLRegisterService` function, which was provided in version 1.0.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`NSLCore.h`

NSLStartNeighborhoodLookup

Starts a search for neighborhoods.

```
NSLError NSLStartNeighborhoodLookup (
NSLRequestRef ref,
NSLNeighborhood neighborhood,
NSLClientAsyncInfo *asyncInfo);
```

Parameters

ref

On input, a value of type `NSLRequestRef` (page 55) created by previously calling `NSLPrepareRequest` (page 38).

neighborhood

On input, `NULL` or an `NSLNeighborhood` (page 54) value created by previously calling `NSLMakeNewNeighborhood` (page 36). Passing `NULL` or a `neighborhood` parameter that was created with a `name` parameter whose value was `NULL` causes this function to get the default neighborhoods. If `neighborhood` was created with a value of `name` that is the name of a neighborhood, this function gets names related to that neighborhood. For example, if `neighborhood` was created with `name` set to "apple.com", this function gets the subdomains of apple.com.

asyncInfo

On input, a pointer to an `NSLClientAsyncInfo` (page 51) structure. Neighborhoods that are found will be stored in the structure's `resultBuffer` field.

function result

An `NSLError` (page 53) structure whose `theErr` field contains the result code. If the value of `NSLError.theErr` is `noErr`, this function successfully started a neighborhood search. Possible errors are `kNSLNotInitialized`, `kNSLSearchAlreadyInProgress`, `kNSLNoPluginsForSearch`, `kNSLBufferTooSmallForData`, and `kNSLNullNeighborhoodPtr`. Call `NSLErrorToString` (page 27) to get a problem and a solution string.

Discussion

This function starts a search for neighborhoods that your application can use to define the scope of a subsequent service search. The result of the search is returned to your application in the `resultBuffer` field of the `NSLClientAsyncInfo` structure pointed to by the `asyncInfo` parameter when the Network Services Location Manager calls your application's notification routine. Call `NSLGetNextNeighborhood` (page 31) to process the data in the result buffer. To cancel an ongoing search, call `NSLCancelRequest` (page 23). To delete a search request reference, call `NSLDeleteRequest` (page 26).

When the Network Services Location Manager calls your application's notification routine, it should check the value of the `searchState` field of the `NSLClientAsyncInfo` structure pointed to by the `asyncInfo` parameter. The `searchState` field will contain one of these values: `kNSLSearchStateBufferFull`, `kNSLSearchStateOnGoing`, `kNSLSearchStateComplete`, or `kNSLSearchStateStalled`.

If the value of the searchState field is `kNSLSearchStatausBufferFull`, your application's notification routine should process the data returned in the `resultBuffer` field of the `NSLClientAsyncInfo` structure and return.

If the value of the `searchState` field is `kNSLSearchStateOnGoing`, the threshold set by the value of the `alertInterval` field or the `alertThreshold` field of the `NSLClientAsyncInfo` structure has been reached. Your application's notification routine should process the data returned in the `resultBuffer` field of the `NSLClientAsyncInfo` structure and return.

If the value of the `searchState` field is `kNSLSearchStateComplete` and the value of `NSLError.theErr` returned by `NSLStartNeighborhoodLookup` is `noErr`, the search is complete. Your application's notification routine should process the data returned in `resultBuffer` and return. If the value of the `searchState` field is `kNSLSearchStateComplete` and the value of `NSLError.theErr` returned by `NSLStartNeighborhoodLookup` is not `noErr`, the error is a fatal error and the search has ended.

If the value of the `searchState` field is `kNSLSearchStateStalled`, the threshold set by the value of the `alertInterval` field or the `maxSearchTime` field of the `NSLClientAsyncInfo` structure has been reached and there is no data in the result buffer. The Network Services Location Manager plug-ins that are processing this neighborhood search request are waiting to receive data from a server and may eventually time out. You can cancel the request by calling `NSLCancelRequest` (page 23).

If the value of `NSLError.theErr` returned by this function is `kNSLBufferTooSmallForData`, the value of the `maxBufferSize` field of the `NSLClientAsyncInfo` structure is too small to hold data that would otherwise have been returned. Your notification routine can cancel the search, modify the size of the buffer, and restart the search, or it can ignore the error and return, thereby resuming the search even though some data has been lost.

Special Considerations

For any request reference, only one neighborhood or service search can be in progress at any one time.

If this function returns an error, the `resultBuffer` field of the `NSLClientAsyncInfo` structure may still contain valid data that was collected before the error occurred. Your notification routine should process the data in the result buffer and return, thereby continuing the search.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`NSLCore.h`

NSLStartServicesLookup

Starts a search for services.

```
NSLError NSLStartServicesLookup (
NSLRequestRef ref,
NSLNeighborhood neighborhood,
NSLTypedDataPtr requestData,
NSLClientAsyncInfo *asyncInfo);
```

Parameters

ref

On input, a value of type NSLRequestRef (page 55) containing a request reference obtained by previously calling NSLPrepareRequest (page 38).

neighborhood

On input, NULL or an NSLNeighborhood (page 54) value created by previously calling NSLMakeNewNeighborhood (page 36). Passing NULL or a neighborhood created with a name parameter whose value was NULL causes this function to start a search for top-level services.

requestData

On input, a value of type NSLTypedDataPtr (page 56) that describes the search parameters. To format requestData properly, call NSLMakeServicesRequestPB (page 37).

asyncInfo

On input, a pointer to a NSLClientAsyncInfo (page 51) structure obtained by previously calling NSLPrepareRequest.

function result

An NSLError (page 53) structure whose theErr field contains the result code. If the value of NSLError.theErr is noErr, this function successfully started a service search. Other possible values are kNSLNotInitialized, kNSLSearchAlreadyInProgress, kNSLNoPluginsForSearch, kNSLNullNeighborhoodPtr, and kNSLBufferTooSmallForData. Call NSLErrorToString (page 27) to get a problem and a solution string.

Discussion

This function starts a service search. The result of the search is returned to your application in the resultBuffer field of the NSLClientAsyncInfo structure pointed to by the asyncInfo parameter when the Network Services Location Manager calls your application's notification routine. Call NSLGetNextURL (page 32) to process the data in the result buffer. To cancel an ongoing search, call NSLCancelRequest (page 23). To delete a search request reference, call NSLDeleteRequest (page 26).

When the Network Services Location Manager calls your application's notification routine, it should check the value of the searchState field of the NSLClientAsyncInfo structure pointed to by the asyncInfo parameter. The searchState field will contain one of these values: kNSLSearchStateBufferFull, kNSLSearchStateOnGoing, kNSLSearchStateComplete, or kNSLSearchStateStalled.

If the value of the searchState field is kNSLSearchStatausBufferFull, your application's notification routine should process the data in the resultBuffer field of the NSLClientAsyncInfo structure and return.

If the value of the searchState field is kNSLSearchStateOnGoing, the threshold set by the value of the alertInterval field or the alertThreshold field of the NSLClientAsyncInfo structure has been reached. Your application's notification routine should process the data in the resultBuffer field and return.

If the value of the searchState field is kNSLSearchStateComplete and the value of NSLError.theErr returned by NSLStartServicesLookup is noErr, the search is complete. Your application's notification routine should process the data returned in resultBuffer and return. If the value of the searchState field is kNSLSearchStateComplete and the value of NSLError.theErr returned by NSLStartServicesLookup is not noErr, the error is a fatal error and the search has ended.

If the value of the `searchState` field is `kNSLSearchStateStalled`, the threshold set by the value of the `alertInterval` field or the `maxSearchTime` field of the `NSLClientAsyncInfo` structure has been reached and there is no data in the result buffer. The Network Services Location Manager plug-ins that are processing this request are waiting to receive data from a server and may eventually time out. You can cancel the request by calling `NSLCancelRequest` (page 23). If the value of the `searchState` field is `noErr`, your application's notification routine should ignore the notification and return immediately, thereby resuming the search.

If the value of `NSLError.theErr` returned by this function is `kNSLBufferTooSmallForData`, the value of the `maxBufferSize` field of the `NSLClientAsyncInfo` structure is too small to hold data that would otherwise have been placed in the result buffer. Your application's notification routine can cancel the search, modify the size of the buffer, and start the lookup again, or it can ignore the error and return, thereby resuming the search even though some data has been lost.

Special Considerations

For any request reference, only one neighborhood or service search can be ongoing at any one time.

If this function returns an error, the `resultBuffer` field of the `NSLClientAsyncInfo` structure may still contain valid data that was collected before the error occurred. Your notification routine should process the data in the result buffer and return, thereby continuing the search.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`NSLCore.h`

Network Services Location Manager Callback Functions

NSLEventProcPtr

Defines a pointer to a system event callback function.

```
typedef void (*NSLEventFilterProcPtr)(
EventRecord *newEvent,
void *userContext);
```

If you name your function `MyNSLEventProcPtr`, you would declare it like this:

```
void MyNSLEventProcPtr (
EventRecord *newEvent,
void *userContext);
```

Parameters

newEvent

A pointer to a structure of type `EventRecord` that describes the event that triggered the execution of this callback function.

displayString

An untyped pointer to arbitrary data that your application previously passed to `NSLStandardGetURL` (page 41).

Discussion

When calling `NSLStandardGetURL`, you can provide an `NSLEventProcPtr` callback function for receiving system events while `NSLStandardGetURL` displays its dialog box. Your system event callback function will receive events while the `NSLStandardGetURL` dialog box is being displayed. If you do not provide an `NSLEventProcPtr` callback function, your application will not receive events.

When your `NSLEventProcPtr` callback function is called, it should process the event immediately and return.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSL.h`

NSLFilterProcPtr

Defines a pointer to a filter callback function. Your filter callback function determines if a URL should be displayed in the `NSLStandardGetURL` dialog box.

```
typedef Boolean (*NSLURLFilterProcPtr)(
char *url,
Str255 displayString);
```

If you name your function `MyNSLFilterProcPtr`, you would declare it like this:

```
Boolean MyNSLURLFilterProcPtr (
char* url,
Str255 displayString);
```

Parameters

url

A pointer to a string containing the URL to filter. Any data in a URL that follows the tag "?NAME=" has been removed from the URL.

displayString

A value of type `Str255`. If your filter callback returns `TRUE`, the URL pointed to by `url` will be listed in the dialog box displayed by `NSLStandardGetURL` (page 41). If you want to change the text of the URL, set `displayString` to the value you want.

Discussion

When calling `NSLStandardGetURL`, you can provide an `NSLURLFilterProcPtr` callback function for filtering the search results before they are displayed in the `NSLStandardGetURL` dialog box. Only those URLs for which your `NSLURLFilterProcPtr` callback function returns `TRUE` will appear in the right column of the dialog box displayed by `NSLStandardGetURL`. You can use the `displayString` parameter to modify the text of each URL that appears in the `NSLStandardGetURL` dialog box.

Network Services Location Manager Data Types

NSLClientAsyncInfo

A structure provided when an application creates a search request and looks for neighborhoods and services.

```
struct NSLClientAsyncInfo {
    void*           clientContextPtr;
    void*           mgrContextPtr;
    char*           resultBuffer;
    long            bufferLen;
    long            maxBufferSize;
    UInt32          startTime;
    UInt32          intStartTime;
    UInt32          maxSearchTime;
    UInt32          alertInterval;
    UInt32          totalItems;
    UInt32          alertThreshold;
    NSLSearchState  searchState;
    NSLError        searchResult;
    NSLEventCode    searchDataType
};
typedef struct NSLClientAsyncInfo NSLClientAsyncInfo;
typedef NSLClientAsyncInfo * NSLClientAsyncInfoPtr;
```

Fields

`clientContextPtr`
A value set by the application for its own use.

`mgrContextPtr`
A value set by the Network Services Location Manager for its own use.

`resultBuffer`
A pointer to the buffer that contains search results when your application's notification routine is called.

`bufferLen`
The number of bytes in `resultBuffer` that contain valid data.

`maxBufferSize`
The length of `resultBuffer`.

`startTime`
Used by the Network Services Location Manager for internal purposes. Your application should not modify this field.

`intStartTime`
Used by the Network Services Location Manager for internal purposes. Your application should not modify this field.

`maxSearchTime`
An application-specified limit in ticks on the total amount of time that is to be expended on the search. The default value is zero, which indicates that the search time is not to be limited. The value of `maxSearchTime` does not override any limit that Mac OS X may impose.

`alertInterval`
: An application-specified value that defines in ticks the interval at which the application's notification routine is to be called. The default value is zero, which indicates that no alert interval is specified.

`totalItems`
: The total number of items in `resultBuffer`.

`alertThreshold`
: An application-specified value that causes the application's notification routine to be called whenever the specified number of items have been placed in `resultBuffer`. Typically, applications set `alertThreshold` to 1.

`searchState`
: A value that describes the current search state. The value can be one of the following: `kNSLSearchStateBufferFull`, `kNSLSearchStateOnGoing`, `kNSLSearchStateComplete`, or `kNSLSearchStateStalled`.

`searchResult`
: An `NSLError` (page 53) structure containing an error code that may have been returned.

`searchDataType`
: An event code that indicates whether the information stored in this `NSLClientAsyncInfo` structure pertains to a neighborhood search (`kNSLNeighborhoodLookupDataEvent`) or a service search (`kNSLServicesLookupDataEvent`).

Discussion
This structure contains information about how a neighborhood or a service search is to be conducted and where search results are to be stored. You obtain a pointer to an `NSLClientAsyncInfo` structure by calling `NSLPrepareRequest` (page 38), and you pass that pointer as a parameter when you call `NSLStartNeighborhoodLookup` (page 44), or `NSLStartServicesLookup` (page 45).

Before calling `NSLStartNeighborhoodLookup` or `NSLStartServicesLookup`, you can modify the way in which the search is conducted by changing certain values in the `NSLClientAsyncInfo` structure. However, once you call `NSLStartNeighborhoodLookup` or `NSLStartServicesLookup`, you should not modify the `NSLClientAsyncInfo` structure.

When `NSLStartNeighborhoodLookup` or `NSLStartServicesLookup` returns, or when your application's notification routine is called, the `NSLClientAsyncInfo` structure contains information about the status of the search and any search results.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSLCore.h`

NSLDialogOptions

The structure returned by `NSLGetDefaultDialogOptions` containing information that controls the appearance of the `NSLStandardGetURL` dialog box.

```
struct NSLDialogOptions {
    UInt16 version;
    NSLDialogOptionFlags dialogOptionFlags;
    Str255 windowTitle;
    Str255 actionButtonLabel;
    Str255 cancelButtonLabel;
    Str255 message;
};
typedef struct NSLDialogOptions NSLDialogOptions;
```

Fields

`version`

A value that specifies the version of the default values.

`dialogOptionFlags`

A bit map. Use the constants defined by the `NSLDialogOptionsFlags` (page 57) enumeration to indicate that the default values should be used or to indicate that the Address text field should not be displayed in the `NSLStandardGetURL` dialog box.

`windowTitle`

A string containing the name that is to appear in the title bar of the dialog box displayed by `NSLStandardGetURL`. The default title is "Connect to Server."

`actionButtonLabel`

A string containing the name that is to be used as the label for the cancel button. If `NULL`, the default name is used. The default name is "Cancel."

`message`

A string containing instructional text that is displayed at the top of the dialog box. If `message` is `NULL`, no instructional text is displayed. The default message is "Choose a server from the list or enter a server address."

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NSL.h`

NSLError

A structure returned by many Network Services Location Manager functions containing a result code as well as a context value.

```
typedef struct NSLError {
    OSStatus    theErr;
    UInt32  theContext;
};
typedef struct NSLError NSLError;
typedef NSLError * NSLErrorPtr;
```

Fields

`theErr`

The result code.

`theContext`

A value the Network Services Location Manager uses to determine the context of an error so that it can provide an appropriate error message when `NSLErrorToString` (page 27) is called.

Discussion
To determine if an error occurred, compare the constant `kNSLErrorNoErr` to the value returned by a function that returns an `NSLError` structure.

If you want to display information about the error to the user, your application should call `NSLErrorToString` (page 27) and pass the `NSLError` structure that your application received. The `NSLErrorToString` function returns two strings — a problem string and a solution string — that your application can display.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSLCore.h`

NSLClientRef

A reference returned by `NSLOpenNavigationAPI` when you open a Network Services Location Manager session.

```
typedef UInt32 NSLClientRef;
```

Discussion
When calling `NSLPrepareRequest` and `NSLCloseNavigationAPI`, pass the client reference that `NSLOpenNavigationAPI` returned.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSLCore.h`

NSLNeighborhood

A value that identifies a neighborhood.

```
typedef unsigned char * NSLNeighborhood;
```

Discussion
A value of type `NSLNeighborhood` is returned by `NSLMakeNewNeighborhood` (page 36) that is used when calling `NSLStartNeighborhoodLookup`, `NSLStartServicesLookup`, `NSLStandardRegisterURL`, and `NSLStandardDeregisterURL`. Call `NSLFreeNeighborhood` (page 28) to deallocate the memory associated with a value of type `NSLNeighborhood`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSLCore.h`

NSLPath

A value used to specify a URL.

```
typedef char * NSLPath;
```

Discussion
A value of type `NSLPath` is used when calling `NSLStandardRegisterURL`, and `NSLStandardDeregisterURL`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSLCore.h`

NSLRequestRef

A request reference returned by `NSLPrepareRequest`.

```
typedef UInt32 NSLRequestRef;
```

Discussion
When calling `NSLStartServicesLookup` or `NSLStartNeighborhoodLookup`, pass the `NSLRequestRef` that `NSLPrepareRequest` returned.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSLCore.h`

NSLServiceType

A value that identifies a service.

```
typedef char * NSLServiceType;
```

Discussion
A value of type `NSLServiceType` is a null-terminated string containing the name of the service, such as `http` or `ftp`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSLCore.h`

NSLServicesList

A value that contains a list of services.

```
typedef Handle NSLServicesList;
```

Discussion
A value of type `NSLServiceList` contains a list of service names.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSLCore.h`

NSLTypedDataPtr

A value created by `NSLMakeServicesRequestPB`.

```
struct NSLTypedData {
unsigned long dataType;
unsigned long lengthOfData;
};
typedef struct NSLTypedData NSLTypedData;
typedef NSLTypedData *NSLTypedDataPtr;
```

Discussion
A value of type `NSLTypedDataPtr` is passed to `NSLStartServicesLookup`. When you no longer need a value of type `NSLTypedDataPtr`, call `NSLFreeTypedDataPtr`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NSLCore.h`

Network Services Location Manager Constants

NSLDialogOptionsFlags

Constants defined for use with the `NSLDialogOptions` structure.

```
enum {
    kNSLDefaultNSLDlogOptions = 0x00000000,
    kNSLNoURLTEField = 0x00000001
};
typedef UInt32 NSLDialogOptionFlags;
```

Constants

`kNSLDefaultNSLDlogOptions`

Use the default values.

Available in Mac OS X v10.0 and later.

Declared in `NSL.h`.

`kNSLNoURLTEField`

Do not display the Address text field.

Available in Mac OS X v10.0 and later.

Declared in `NSL.h`.

Discussion

The `NSLDialogOptionFlags` enumeration defines constants that tell the `NSLStandardGetURL` (page 41) function how to display its dialog box.

NSLEventCode

Constants defined for use in the `searchDataType` field of the `NSLClientAsyncInfo` structure.

```
enum {
    kNSLServicesLookupDataEvent = 6,
    kNSLNeighborhoodLookupDataEvent = 7,
    kNSLSNewDataEvent = 8,
    kNSLContinueLookupEvent = 9
};
typedef UInt16 NSLEventCode;
```

Constants

`kNSLServicesLookupDataEvent`

A search for services has caused your notification routine to be called.

Available in Mac OS X v10.0 and later.

Declared in `NSLCore.h`.

`kNSLNeighborhoodLookupDataEvent`
A search for neighborhoods has caused your notification routine to be called.

Available in Mac OS X v10.0 and later.

Declared in `NSLCore.h`.

`kNSLSNewDataEvent`
`kNSLContinueLookupEvent`
Reserved.

Available in Mac OS X v10.0 and later.

Declared in `NSLCore.h`.

NSLSearchState

Constants defined for use in the `searchState` field of the `NSLClientAsyncInfo` structure to describe the state of a search.

```
enum {
    kNSLSearchStateBufferFull = 1,
    kNSLSearchStateOnGoing = 2,
    kNSLSearchStateComplete = 3,
    kNSLSearchStateStalled = 4,
    kNSLWaitingForContinue = 5
};
typedef UInt16 NSLSearchState;
```

Constants

`kNSLSearchStateBufferFull`
The result buffer is full.

Available in Mac OS X v10.0 and later.

Declared in `NSLCore.h`.

`kNSLSearchStateOnGoing`
The search is still in progress and the result buffer may contain data.

Available in Mac OS X v10.0 and later.

Declared in `NSLCore.h`.

`kNSLSearchStateComplete`
The search is complete and the result buffer may contain data.

Available in Mac OS X v10.0 and later.

Declared in `NSLCore.h`.

`kNSLSearchStateStalled`
The search is still in progress, but there is no data in the result buffer; the search may eventually time out without returning any results.

Available in Mac OS X v10.0 and later.

Declared in `NSLCore.h`.

`kNSLWaitingForContinue`
Obsolete.

Available in Mac OS X v10.0 and later.

Declared in `NSLCore.h`.

Discussion
The `NSLSearchState` enumeration defines constants that describe the state of a search.

Network Services Location Manager Result Codes

The result codes specific to the Network Services Location Manager are listed here. In addition, Network Services Location Manager functions may return other Mac OS X result codes, which are described in *Inside Mac OS X*. Some result codes cannot be returned by this version of the Network Services Location Manager, as noted.

Result Code	Value	Description
`noErr`	0	No error. Available in Mac OS X v10.0 and later.
`kNSLSchedulerError`	–4171	A custom thread routine encountered an error. Available in Mac OS X v10.0 and later.
`kNSLBadURLSyntax`	–4172	The caller passed a URL that contains characters that are not allowed. Available in Mac OS X v10.0 and later.
`kNSLUILibraryNotAvailable`	–4174	The Network Services Location Manager UI Library is not available; this result code cannot be returned by this version of the Network Services Location Manager. Available in Mac OS X v10.0 and later.
`kNSLErrNullPtrError`	–4176	A specified pointer is invalid. Available in Mac OS X v10.0 and later.
`kNSLSomePluginsFailedToLoad`	–4177	During system initialization, the Network Services Location Manager was unable to load some plug-ins; this result code cannot be returned by this version of the Network Services Location Manager. Available in Mac OS X v10.0 and later.
`kNSLNullNeighborhoodPtr`	–4178	The pointer to a neighborhood is invalid. Available in Mac OS X v10.0 and later.
`kNSLNoPluginsForSearch`	–4179	None of the installed plug-ins are able can respond to the search request; this result code cannot be returned by this version of the Network Services Location Manager. Available in Mac OS X v10.0 and later.
`kNSLSearchAlreadyInProgress`	–4180	A search is already in progress for the specified value of type `NSLClientRef`. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kNSLNoPluginsFound	–4181	During system initialization, the Network Services Location Manager was unable to find any valid plug-ins to load; this result code cannot be returned by this version of the Network Services Location Manager. Available in Mac OS X v10.0 and later.
kNSLPluginLoadFailed	–4182	During system initialization, the Network Services Location Manager was unable to load one of the plug-ins; this result code cannot be returned by this version of the Network Services Location Manager. Available in Mac OS X v10.0 and later.
kNSLBadProtocolTypeErr	–4183	The specified value of type NSLServiceType is empty. Available in Mac OS X v10.0 and later.
kNSLNullListPtr	–4184	The pointer to the specified list is invalid. Available in Mac OS X v10.0 and later.
kNSLBadClientInfoPtr	–4185	The specified value of type NSLClientAsyncInfoPtr is invalid. Available in Mac OS X v10.0 and later.
kNSLCannotContinueLookup	–4186	The search cannot be continued due to an error condition or a bad state. Available in Mac OS X v10.0 and later.
kNSLBufferTooSmallForData	–4187	The application's result buffer is too small to store the data returned by a plug-in. Available in Mac OS X v10.0 and later.
kNSLNoContextAvailable	–4188	A parameter of type NSLClientAsyncInfo is invalid. Available in Mac OS X v10.0 and later.
kNSLRequestBufferAlreadyInList	–4189	Reserved. Available in Mac OS X v10.0 and later.
kNSLInvalidPluginSpec	–4190	The theContext field of the specified NSLError (page 53) structure is invalid. Available in Mac OS X v10.0 and later.
kNSLNoSupportForService	–4191	No plug-in supports the requested service registration or deregistration; this result code cannot be returned by this version of the Network Services Location Manager. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`kNSLBadNetConnection`	–4192	A network error occurred. The computer may not be connected to the network. Available in Mac OS X v10.0 and later.
`kNSLBadDataTypeErr`	–4193	The specified parameter is not of the correct data type. Available in Mac OS X v10.0 and later.
`kNSLBadServiceTypeErr`	–4194	The specified service type is not supported. Available in Mac OS X v10.0 and later.
`kNSLBadReferenceErr`	–4195	The specified value of type `NSLClientRef` or `NSLRequestRef` is invalid. Available in Mac OS X v10.0 and later.
`kNSLNoElementsInList`	–4196	A specified list is empty. Available in Mac OS X v10.0 and later.
`kNSLNotInitialized`	–4199	The Network Services Location Manager could not be initialized. Available in Mac OS X v10.0 and later.

Document Revision History

This table describes the changes to *Network Services Location Manager (Legacy)*.

Date	Notes
2006-05-23	Added information about deprecation.

Document Revision History