Network Services Location Manager 1.1 Developer's Kit

Technical Publications

Contents

Figures, Tables, and Listings

About This Manual

This document describes the programming interface for the Network Services Location (NSL) Manager 1.1. The NSL Manager provides a protocol-independent way for applications to discover available network services with minimal network traffic.

Changes Since NSL 1.0

The following list summarizes the differences between NSL 1.0 and NSL 1.1:

- The `ClientAsyncInfo` structure has been renamed to the `NSLClientAsyncInfo` structure. This change affects NSL Manager functions that take this structure as a parameter.
- Functions that took parameters of type `NSLClientNotifyProcPtr` now take parameters of type `NSLClientNotifyUPP`.
- The NSL Manager now provides two functions for verifying the presence of the NSL Manager and its version: `NSLLibraryPresent` and `NSLLibraryVersion`.
- The NSL Manager now provides a function, `NSLStandardGetURL`, that displays a dialog box that allows the user to search for services. The new function, `NSLGetDefaultDialogOptions`, is a utility function for controlling the appearance of the dialog box. A new structure, `NSLDialogOptions`, is used to store values that control the appearance of the dialog box.
- The `NSLRegisterService` and `NSLDeregister` service functions have been renamed to `NSLStandardRegisterURL` and `NSLStandardDeregisterURL`, respectively. The parameters to these functions have been modified so that they can be called without previously calling `NSLOpenNavigationAPI`.
- The NSL Manager now provides an `NSLNewThread` function and an `NSLDisposeThread` function for plug-ins to call, instead of calling the Thread Manager's `NewThread` and `DisposeThread` functions. The NSL Manager also provides a new utility function (`NSLCopyNeighborhood`) that plug-ins can call to copy NSL neighborhoods.

- The NSL Manager now provides new utility functions for getting the name from a neighborhood (`NSLGetNameFromNeighborhood`), getting the length of a neighborhood (`NSLGetNeighborhoodLength`), getting the service portion of a URL (`NSLGetServiceFromURL`), encoding portions of a URL (`NSLHexEncodeText`), decoding portions of a URL, and determining whether a service is in a services list (`NSLGetServiceFromURL`).
- The `NSLMakeRegistrationPB` has been removed. It is made obsolete by the new `NSLStandardRegisterService` function.
- The `NSLMakeRequestPB` had been renamed to `NSLMakeServicesRequestPB` and its parameters have been modified.

Conventions Used in This Manual

The Courier font is used to indicate function names, code, and text that you type. This manual includes special text elements to highlight important or supplemental information:

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

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

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

For more information

The following sources provide additional information that is important for NSL developers:

- *Inside Macintosh*, available online at http://devworld.apple.com/techinfo/techdocs/mac/mac.html
- *NSL Network Administrators Guide*, which tells administrators how to configure DNS and SLP servers so they can participate in NSL lookups (available with the final version of the NSL SDK)
- *DNS and Bind* by Paul Albitz and Cricket Liu, O'Reilly & Associates, Inc. 1994
- RFC 2589, *Lightweight Directory Access Protocol Version 3*, available at ftp://ftp.isi.edu/in-notes/rfc2589.txt.
- RFC 2608, *Service Location Protocol Version 2*, available at http://www.rfc-editor.org/rfc/rfc2608.txt.

C H A P T E R 1

About the Network Services Location Manager

The Network Services Location (NSL) Manager provides a protocol-independent way for applications to discover available network services with minimal network traffic.

The NSL Manager provides

- AppleTalk-like ease-of-use for the dynamic discovery of 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 will become easier to use when they call the NSL Manager. For example,

- Instead of requiring the user to type a URL to locate a web server, a browser application that calls the NSL 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 NSL 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.

This chapter describes how you can use the NSL Manager to

- add network-service search functionality to your application

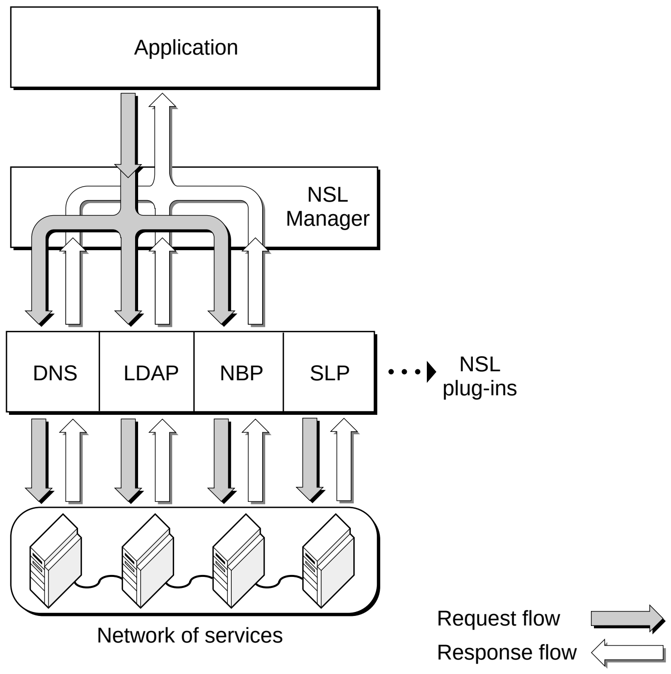
- register a network service with the NSL Manager so that it can be found in searches

Version 1.1 of the NSL Manager runs only on Power PC computers on which Mac OS 9 is installed. Before your application calls the NSL Manager, it should verify that Mac OS 9 is running.

Note
The NSL Manager calls the Thread Manager, so applications that cannot call the Thread Manager should not call the NSL Manager. In addition, mixed-mode applications should not call the NSL Manager. ◆

This version of the NSL Manager comes with plug-ins for the following protocols: Domain Name Service (DNS), Service Location Protocol (SLP), Name Binding Protocol (NBP), and Lightweight Directory Access Protocol (LDAP). Figure 1-1 illustrates the relationship between applications, the NSL Manager, and the NSL plug-ins.

Figure 1-1 Flow of a network service lookup



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

- a services list, which is an NSL data type that specifies the services that are to be searched for.
- a neighborhood, which is an NSL data type that represents some part of a network and the protocols that are relevant to that part. Table 1-1 lists some examples of neighborhoods.

Table 1-1 Examples of neighborhoods

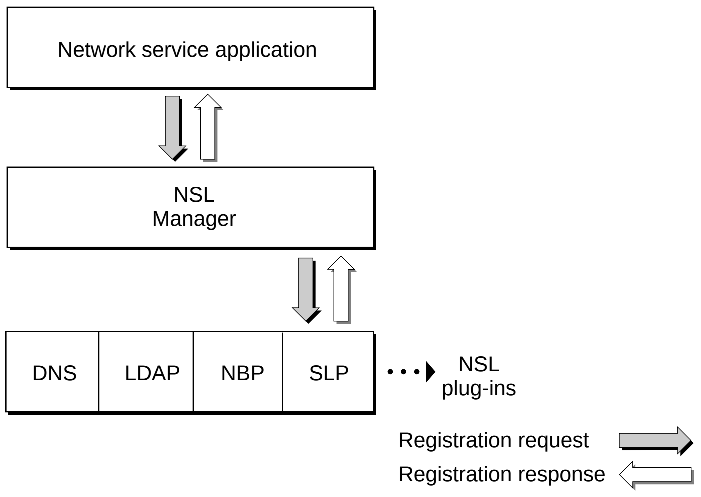
Network Portion	**Protocol**
apple.com	DNS, SLP, and LDAP
My AppleTalk Zone	NBP
Local Service	SLP

The following steps outline the flow of a service lookup:

1. The application creates a lookup request and calls the NSL Manager's `NSLStartServicesLookup` function.
2. The NSL Manager receives the request and passes it to those NSL plug-ins that are capable of responding to the request.
3. Each NSL plug-in that receives the request starts to look for the specified services.
4. Providers of services send their responses to the NSL plug-ins.
5. The NSL plug-ins pass the responses to the NSL Manager.
6. The NSL Manager passes the responses to the application that initiated the lookup. If more than one plug-in responds, the NSL Manager returns the responses to the application in a single response stream.

Applications that provide services can register themselves with the NSL Manager as shown in Figure 1-2.

Figure 1-2 Flow of an SLP service registration



The following steps outline the flow of a service registration:

1. The application creates a value of type `NSLPath` that specifies the URL to register, calls `NSLHexEncodeText` to encode any illegal characters that may be in the URL, and calls the NSL Manager's `NSLStandardRegisterURL` function to register the URL in a specific neighborhood.
2. The NSL Manager receives the request and passes it to the NSL plug-ins that are capable of registering the service.
3. The NSL plug-in receives the request and registers the service.
4. The NSL Manager returns a value to the application indicating that the service was registered successfully.

About NSL Plug-ins

An NSL plug-in is an extension that searches for services. It makes itself available to the NSL Manager when the NSL Manager is initialized, and it resides in memory only when it is responding to lookup requests from applications.

Note
The Extensions Manager can be used to enable and disable individual NSL plug-ins. ◆

The NSL Manager can pass lookup requests to any plug-in that adheres to the NSL Manager API.

When the NSL Manager is initialized, each NSL plug-in provides the following information to the NSL Manager:

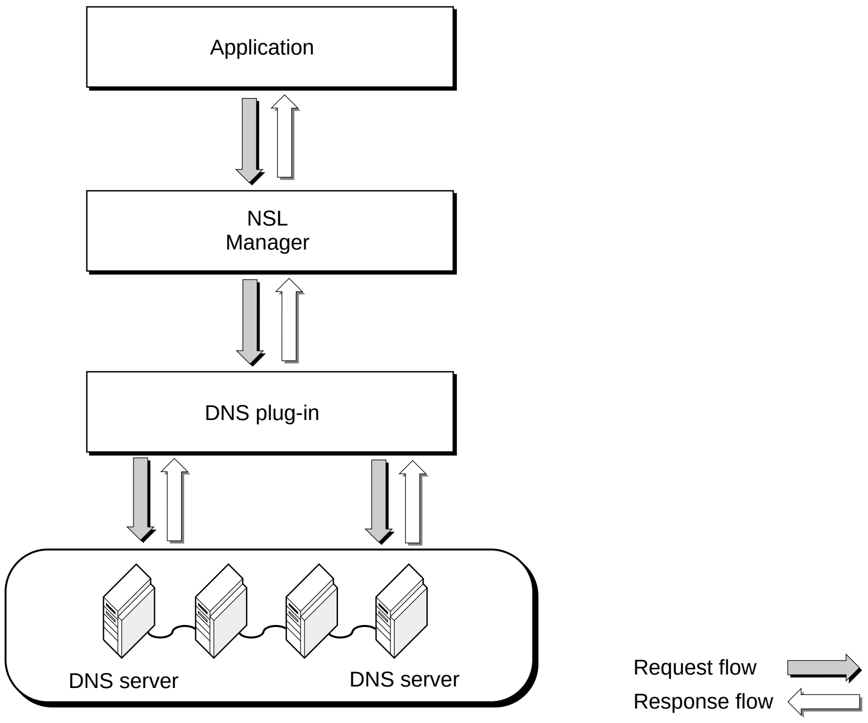
- the types of services the plug-in can search for, such as HTTP
- the protocol the plug-in uses to conduct searches, such as DNS

The NSL SDK comes with two NSL plug-ins: DNS and SLP.

About the DNS Plug-in

The DNS plug-in allows applications to receive lists of services from DNS servers. The information about each service is taken from the TXT record for each domain for which the server is responsible. Figure 1-3 shows the flow of a DNS lookup.

Figure 1-3 Flow of a DNS lookup



The DNS plug-in provides the following routines for the NSL Manager to call:

- An initialization routine that allocates memory and opens network connections to DNS servers
- A deinitialization routine that deallocates memory and closes network connections
- A start-neighborhood-lookup routine that starts a neighborhood lookup
- A start-services-lookup routine that starts a service lookup
- A continue-lookup routine that resumes a lookup for services or neighborhoods that has paused in order to deliver lookup results to the application
- A cancel-lookup routine that cancels an ongoing lookup

- An error-number conversion routine that provides a pair of strings describing the error and a possible solution for any error number that the plug-in may return
- An information routine that provides details about the services and protocols the plug-in supports, as well as a comment string that describes the services and protocol the plug-in supports

About the SLP Plug-in

The SLP plug-in uses the Service Location Protocol to locate services. The Service Location Protocol is an emerging Internet Engineering Task Force (IETF) protocol designed to simplify the discovery and use of network resources. SLP is well-suited for client-server applications and for establishing connections between network peers that offer or consume generic services. SLP supports servers that register services dynamically as well as clients that use multicast protocols to locate services.

Note
Version 1.1 of Apple Computer's SLP plug-in conforms to SLP Version 2, as described in RFC 2608, and is not compatible with implementations of version 1 of that specificiation nor is it compatible with version 1.0 or version 1.0.1 of Apple Computer's SLP plug-in. ◆

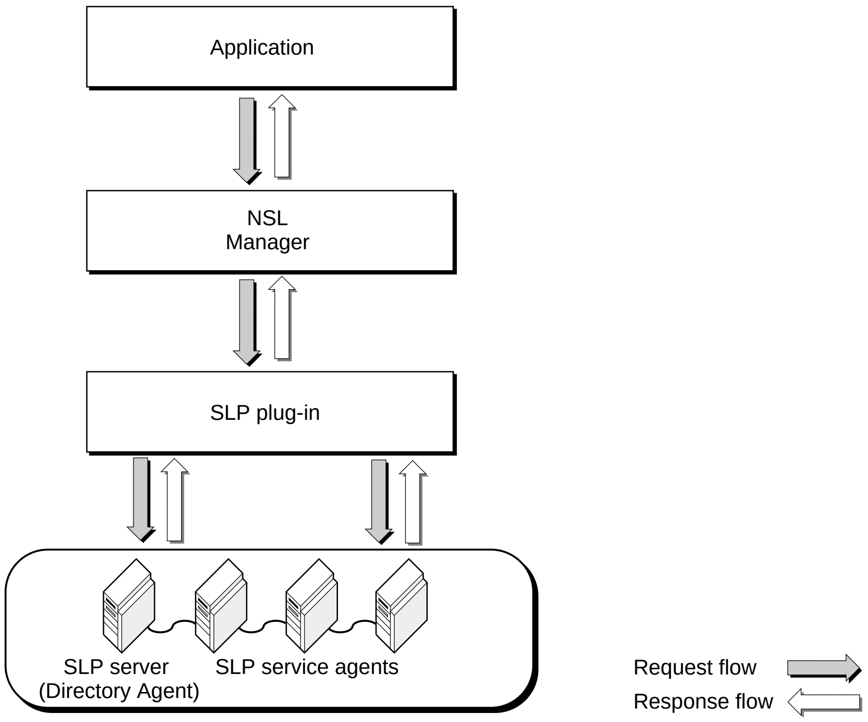
The SLP plug-in accepts service registrations from applications that provide network services running on the local host. When the SLP plug-in registers a service, it creates for that service an SLP Service Agent. Service Agents listen for lookup requests and respond appropriately when the SLP plug-in queries them.

The SLP plug-in also listens for and registers with any SLP Directory Agent Servers (DAs) that may be present on the local subnet. The SLP plug-in then listens for and registers with any other DAs that may announce their availability on the local subnet.

Note
When the SLP plug-in is first loaded into memory, it uses IP multicast to locate DAs. To give the SLP plug-in access to services beyond its immediate subnet, routers on the local subnet must be configured to support IP multicast. ◆

If a network has a DA, Service Agents register themselves with the DA. The SLP plug-in can then query the DA directly, thereby minimizing network traffic. In Figure 1-4, the SLP plug-in can bypass the Service Agents and query the DA directly. If the DA becomes unavailable, the SLP plug-in will query each Service Agent individually.

Figure 1-4 Flow of an SLP lookup



Like the DNS plug-in, the SLP plug-in provides routines that initialize and deinitialize the plug-in, start, continue, and cancel a service or neighborhood lookup, return a pair of strings that describe an error condition and a possible solution for any error code that the SLP plug-in may return, and a routine that returns information that describes the plug-in's capabilities. The SLP plug-in also provides routines to register and deregister services.

When an application registers with the SLP plug-in, it has the option of providing a neighborhood to register in. If the application provides a neighborhood, the SLP plug-in uses the neighborhood as an SLP scope. Subsequent SLP lookups will include that neighborhood.

If the application does not provide a neighborhood, the SLP plug-in determines the scope as follows:

- If the URL is a host name, the SLP plug-in generates the scope using the first search domain name returned from the list of search domains configured in the TCP / IP control panel. For example, the host name "charlie.lucid.apple.com" would result in a neighborhood of "lucid.apple.com" if the first search domain name is "apple.com".
- If the URL is an IP address, the SLP plug-in uses a default neighborhood that is converted to a localized string in the SLP plug-in's resource. In English, the default neighorhood is "Local Services".

For more information about SLP, see RFC 2165.

About the NBP Plug-in

The Network Bind Protocol (NBP) plug-in locates Network Bind Procotol (NBP) services and returns AppleTalk-style URLs describing NBP tuples to applications that look up AppleTalk network services. In AppleTalk terminology, a zone is a neighborhood, and the default neighborhood is the local AppleTalk zone.

The NBP plug-in retrieves a list of all AppleTalk zones when an application requests as a neighborhood lookup on the local zone.

Note
The NBP plug-in does not support the registration of AppleTalk services. ◆

About the LDAP Plug-in

The Lightweight Directory Access Protocol (LDAP) plug-in locates LDAP services. The LDAP plug-in is similar to the DNS plug-in in that it queries servers that have been statically defined in the LDAP control panel.

If the queried LDAP server supports LDAP version 3 or later, the LDAP plug-in returns the containers retrieved from the server as its default neighborhoods. If the queried LDAP server is a version 2 LDAP server, the LDAP plug-in returns all data located located on that server that match the requested service type.

Note
The LDAP plug-in does not support the registration of LDAP services. ◆

Using the Network Services Location Manager

There are two ways to use the NSL Manager to search for network services:

- Your application can call the NSL Manager's `NSLStandardGetURL` function, which displays the "Select a Service" dialog box. The "Select a Service" dialog box allows the user to choose the services to search for and the neighborhoods in which to search. The `NSLStandardGetURL` function creates request parameter blocks and lookup requests based on the user's selections and starts, stops, and continues searches under the user's control. If the user selects a URL from the search results, the `NSLStandardGetURL` function returns that URL to the calling application.
- Your application can call NSL Manager low-level functions that open a session with the NSL Manager, create a request parameter block, create a lookup request, start a lookup, and continue a lookup.

Using the NSL Manager's High-Level Functions

Registering and Deregistering Services

If your application only needs to register and deregister a network service, it can call the following NSL Manager functions without having to call any other NSL Manager functions:

- `NSLHexEncodeText`. This function encodes any illegal characters that may be in the URL so that the URL can be processed by `NSLStandardRegisterURL`. The parameters for calling `NSLHexEncodeText` are

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

On input, the `rawText` parameter contains the portion of the 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 needs to be encoded is `NAME=Kevs/G3`. The forward slash (`/`) in that portion is an illegal character.

On input, the `rawTextLen` parameter contains the length of the URL specified by the `rawText` parameter.

On output, `newTextBuffer` parameter contains the encoded text and the `newTextBufferLen` parameter contains the length of valid data in `newTextBuffer`.

If, on output, the contents of the string pointed to by `rawText` is different from the contents of the string pointed to be `newTextBuffer`, the Boolean value pointed to be textChanged is `TRUE`.

- `NSLStandardRegisterURL`. This 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 name of 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 NSL plug-ins determine the neighborhood in which to register the service.

- `NSLStandardDeregisterURL`. This function deregisters a registered network service.

  The parameters for calling `NSLStandardDeregisterURL` are

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

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

The `neighborhoodToRegisterIn` parameter is `NULL`. When this parameter is `NULL`, the NSL plug-ins determine the neighborhood in which the service is to be deregistered.

Displaying the “Select a Service” Dialog Box

Applications that need to allow the user to specify the services to search for and the neighborhoods in which to search can call the `NSLStandardGetURL` function to display a dialog box that allows the user to specify his or her choices. The `NSLStandardGetURL` function displays the “Select a Service” dialog box.

Note
If you call `NSLStandardGetURL`, you do not need to call NSLOpenNavigationAPI or any other NSL Manager function. The `NSLStandardGetURL` function handles the interface between the calling application and the NSL Manager. ◆

Applications that call `NSLStandardGetURL` specify the service types for which the search is to be conducted. The plug-ins that are capable of participating in the search provide a starting point for the search by providing a list of default neighborhoods. The user can expand the scope of the search by clicking the disclosure triangle next to each neighborhood or service.

If the user selects a URL and clicks the Choose button, `NSLStandardGetURL` returns the selected URL to the calling application and dismisses the “Select a Service” dialog box.

Note
The “Select a Service” dialog box displayed by `NSLStandardGetURL` is a movable modal dialog box. ▲

The parameters for calling `NSLStandardGetURL` is

```
OSStatus NSLStandardGetURL (
                      NSLDialogOptions* dialogOptions,
                      NSLEventUPP eventproc,
                      void* eventProcContextPtr,
                    NSLURLFilterUPP filterproc,
                    char* serviceTypeList,
                    char** urlSelectedURL);
```

The `dialogOptions` parameter is pointer to an `NSLDialogOptions` structure whose fields specify the appearance of the dialog box, such as the text that appears in the title bar of the “Select a Service” dialog box and the text that appears as button labels.

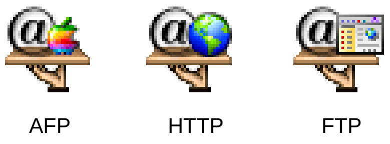
The `eventproc` parameter points to an application-defined system event callback routine that the NSL Manager calls so that your application can handle events that may occur while the “Select a Service” dialog box is displayed. If `eventProc` is `NULL`, your application will not receive update events while the “Select a Service” dialog box is displayed.

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

The `filterproc` parameter is a value of type `NSLURLFilterUPP` that points to an application-defined callback routine that your application can use to filter the results that are displayed.

The `serviceTypeList` is a null-terminated string of tuples that specify the services that are to be search for. You can use the `serviceTypeList` parameter to specify the service icon that is displayed for each service that is found. If you specify a custom icon, your custom icon is displayed. Otherwise, default NSL icons are displayed. Figure 2-1 shows the NSL service icons.

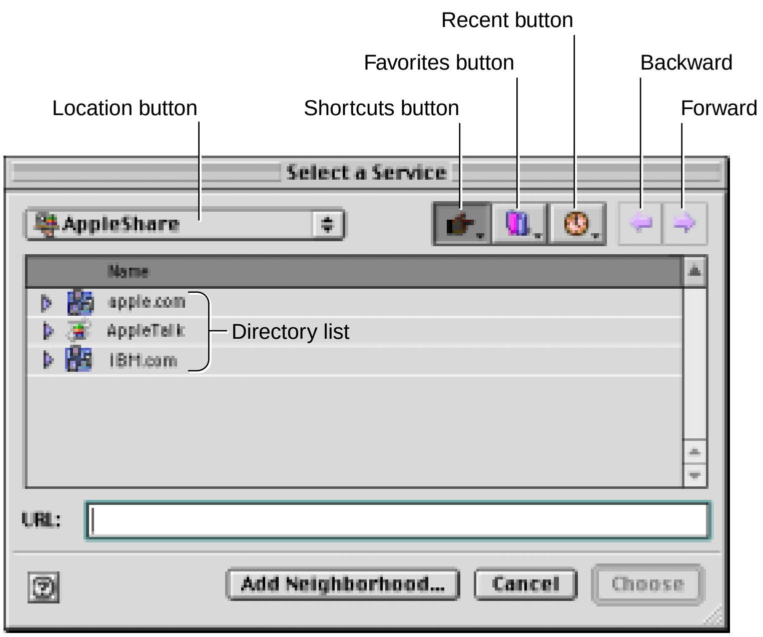
Figure 2-1 Service icons



The `urlSelectedURL` parameter contains the URL the user selected when `NSLStandardGetURL` returns. The `url` parameter is empty if the user dismisses the "Select a Service" dialog box without selecting a URL.

Figure 2-2 shows the "Select a Service" dialog box as it might appear when first displayed. The Directory list consists of the neighborhoods that the plug-ins become aware of when they are first initialized.

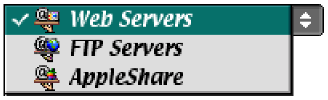
Figure 2-2 The NSL Manager's "Select a Service" dialog box



The elements of the "Select a Service" dialog box are

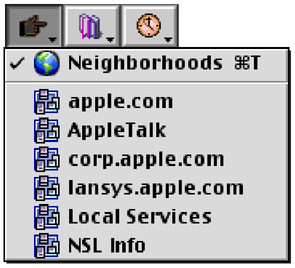
- Services Selection menu. When the dialog box is first displayed, the Services Selection menu displays the currently selected service type as specified by the calling application in the `serviceTypeList` parameter of the `NSLStandardGetURL` function. For example, if the value of `serviceTypeList` is "`http,https;FTP Servers,ftp;AppleShare, afp;Web Servers`", the Shortcuts menu will look the menu shown in Figure 2-4.

Figure 2-3 Services Selection menu



- Shortcuts menu. When the “Select a Service” dialog box is first displayed, the Shortcuts menu lists all of the default neighborhoods returned by all of the NSL plug-ins. To narrow the focus to the neighborhood supported by a particular protocol, the user the user can select one of the protocols listed in the menu. To display the services of all available protocols, the user selects the Neighborhoods item in the Shortcuts menu. Figure 2-4 shows a sample of the Shortcuts menu.

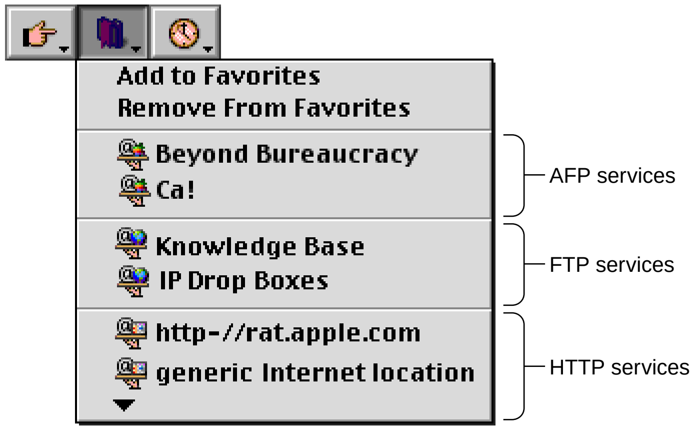
Figure 2-4 Shortcuts menu



- Favorites menu. This pop-up menu lists neighborhoods, services, or a combination of neighborhoods and services that the user saved previously. The services are filtered by the service types in the Shortcuts menu.

  Figure 2-5 shows a sample of the items that might appear in the Favorites menu:

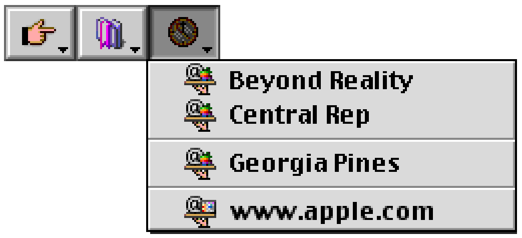
Figure 2-5 Favorites menu



The user saves favorites by selecting one or more neighborhoods or services in the Directory list and choosing the “Add to Favorites” command in the Favorites menu.
The user can remove services from the Favorites list by choosing the “Remove From Favorites” command in the Favorites menu. Choosing “Remove From Favorites” causes a dialog box to appear that lists each favorite neighborhood or service. The user selects the items to remove and clicks the Remove button.

- Recent menu. This pop-up menu shows a list of services recently selected by the user in previous uses of the “Select a Service” dialog. Like the Favorites menu, the services listed in the Recent menu are filtered by the service types listed in the Shortcuts menu. Figure 2-6 shows a sample Recent menu.

Figure 2-6 Recent menu



- Directory list. When the “Select a Service” dialog box first appears, the Directory list consists of the default neighborhoods of the plug-ins that are capable of searching for services specified by the `servicetypeList` parameter. (The default neighborhoods are the neighborhoods that the plug-ins become aware of when they are first initialized.) Thereafter, the list shows the results of any searches.

  Each entry in the Directory list includes an icon provided by the calling application or by the NSL Manager if the calling application does not provide an icon. The icon is followed by the name of the neighborhood or service obtained as a result of a search, with the service information (such as http://) removed. For example, the URL http://www.apple.com would appear as www.apple.com.

  The Directory list supports drag and drop. Items that are dragged from Directory list to the Finder become URL aliases.

  The “Select a Service” dialog can only return one URL to the calling application, so when the user selects more than one item in the Directory list, the Choose button is dimmed.
- Backward arrow. The Backward arrow becomes active when the user double-clicks an item in the Directory list. Clicking the Backward arrow causes the parent of the selected item to be displayed.
- Forward arrow. The Forward arrow becomes available when the user clicks the Backward arrow. Clicking the Forward arrow when a directory in the Directory list is selected causes the children of the selected item to be displayed.
- URL field. This editable text field, whose appearance can be controlled by the calling application, shows the full URL of the selected item in the . The user can append additional path information to the URL or can bypass the search mechanism by entering a complete URL and pressing the Select button.
- Add Neighborhood button. Clicking this button causes a dialog box to appear that allows the user to enter the name of a neighborhood. When the user clicks the OK button in the Add Neighborhood dialog box, the neighborhood that the user entered is added to the Directory list.

The user can start a search in two ways:

- By clicking the disclosure triangle next to a neighborhood. This sends a request to the NSL Manager for a list of neighborhoods associated with this neighborhood and any services located in this neighborhood. The user can click more than one disclosure triangle to initiate additional simultaneous searches. The user can cancel any ongoing search by clicking its disclosure triangle. A search that was started by clicking a disclosure triangle cannot be paused by clicking the Backward button.

- By double-clicking a neighborhood or service. The NSL Manager adds the name of the neighborhood to the Location menu, clears the Directory list, and displays the search results in the Directory list.

The user can stop a search in the following ways:

- By clicking a disclosure triangle if the search was started by clicking that disclosure triangle.
- By clicking the Cancel button.

The NSL Manager displays an alert dialog box for any errors that occur. The NSL Manager calls `NSLErrorToString` so that the alert dialog box can display an error string describing the error and a solution string.

Using the NSL Manager’s Low-Level Functions

Applications that call NSL Manager functions must call standard application initialization functions, such as `MaxApplZone`. Applications must also call `YieldToAnyThread` from their main event loop in addition to calling `WaitNextEvent` or `SystemTask`. If your application does not call `YieldToAnyThread`, service lookups will fail and your application will appear to hang.

To search for network services using the NSL Manager’s low-level functions, an application calls `NSLOpenNavigationAPI` to initialize the NSL Manager, as shown in Listing 2-1.

Listing 2-1 Initializing the NSL Manager

```
OSStatus status;
NSLClientRef myClientRef;
status = NSLOpenNavigationAPI( &myClientRef );
```

The NSL Manager returns a client reference that the application uses to prepare a lookup request and to call `NSLCloseNavigationAPI` when the application no longer needs to make lookup requests.

Next, the application calls `NSLMakeNewServicesList` to create a services list and calls `NSLMakeServicesRequestPB` to convert the resulting services list into a request parameter block, as shown in Listing 2-2.

Listing 2-2 Creating a request parameter block

```
NSLServicesList serviceList = NULL;
serviceList = NSLMakeNewServicesList( "http,ftp" );
iErr.theErr = NSLMakeServicesRequestPB( serviceList, &newDataPtr );
```

In Listing 2-2, the application creates a services list that specifies that HTTP and FTP services are to be searched for. If the application doesn't specify any services, all services will be searched for. The application then calls `NSLMakeServicesRequestPB` with the services list as a parameter. The `NSLMakeServicesRequestPB` function formats the services list in a way that allows any plug-in to parse the services list properly.

Next, the application creates a lookup request by calling `NSLPrepareRequest`, as shown in Listing 2-3.

Listing 2-3 Preparing an NSL lookup request

```
long bufLen = 4096;
char* buffer = NULL;
NSLRequestRef myRequestRef;
NSLClientAsyncInfoPtr myAsyncInfo;
NSLError iErr = kNSLErrorNoErr;

buffer = NewPtr( bufLen );

iErr = NSLPrepareRequest( NULL, NULL, myClientRef, &myRequestRef,
                              buffer, bufLen, &myAsyncInfo );
if ( iErr.theErr )
```

```
{
    // Handle error.
}
```

Calling `NSLPrepareRequest` returns a `requestRef` and sets up an `NSLClientAsyncInfo` structure for this request. The application uses the `NSLClientAsyncInfo` structure to search for neighborhoods and services. The application can control the way the search is conducted by specifying

- a maximum time for the search
- an alert threshold (that is, return search results whenever a certain number if items have been returned)
- an alert interval (that is, return search results whenever a specified time elapses)

The NSL Manager uses the `NSLClientAsyncInfo` structure to convey search results and status information about the search from the plug-in to the application.

In Listing 2-4, the application calls `NSLStartNeighborhoodLookup` to obtain the first available neighborhood on the local network and calls `NSLContinueLookup` until it has obtained all of the available neighborhoods on the local network.

Listing 2-4 Searching for neighborhoods

```
char *name;
long nameLength;
long neighborhoodLength;
NSLNeighborhood neighborhood;

// Set the values of the myAsyncInfo structure
myAsyncInfo->maxSearchTime = 0; // no max search time
myAsyncInfo->alertInterval = 0; // no alert interval
myAsyncInfo->alertThreshold = 1; // return after each item

if ( iErr.theErr == noErr )
    iErr = NSLStartNeighborhoodLookup( myRequestRef, neighborhood,
              myAsyncInfo );
    do {
        if ( iErr.theErr == noErr && myAsyncInfo->totalItems > 0 )
        {
            while ( NSLGetNextNeighborhood( myAsyncInfo, &nhPtr,
                    &neighborhoodLength ) )
            {
                if ( neighborhoodLength > 0 &&
                    neighborhoodLength < kBufferLength  )
                {
                    NSLGetNameFromNeighborhood( &name, &nameLength,
                        neighborhood);
                }
                else
                {
                    done = true;
                }
            }
            if ( myAsyncInfo->searchState == kNSLSearchStateComplete )
                done = true;
        else
            iErr = NSLContinueLookup( myAsyncInfo );
    }

} while ( !iErr.theErr && !done );

    if ( buffer )
        DisposePtr(buffer);
}
```

The application could display the name of each neighborhood and allow the user to select one.

In Listing 2-5, the application calls `NSLStartServicesLookup` to start the service lookup in the selected neighborhood, as specified by the `neighborhood` parameter. The `myRequest` parameter was created earlier by calling `NSLPrepareRequest` and the `newDataPtr` parameter was created earlier by calling `NSLMakeServicesRequestPB`.

The application continues to call `NSLContinueLookup` until it has received information about all of the services that match the search criteria.

Listing 2-5 Searching for services

```
iErr = NSLStartServicesLookup( myRequestRef, neighborhood, newDataPtr,
       myAsyncInfo );
do {
    if ( iErr.theErr == noErr && myAsyncInfo->totalItems > 0 )
    {
        while ( NSLGetNextUrl( myAsyncInfo, &urlPtr, &urlLength ) )
        {
            if ( urlLength > 0 )
                {
                    // Process the result buffer.
                }
            else
            {
                done = true;
            }
        }
        if ( myAsyncInfo->searchState == kNSLSearchStateComplete )
            done = true;
        else
            iErr = NSLContinueLookup( myAsyncInfo );
        }
    } while ( !iErr.theErr && !done );
```

When the lookup is complete, the application reclaims memory allocated for the services list, the request parameter block, and the lookup request, as shown in Listing 2-6.

Listing 2-6 Reclaiming memory

```
NSLDisposeServicesList(serviceList);

// Calling NSLDeleteRequest releases the memory associated with the
// NSLClientAsyncInfo structure.

NSLDeleteRequest(myRequestRef);
NSLFreeTypedDataPtr(newDataPtr);
```

When the application has no need to make additional lookups, it calls `NSLCloseNavigationAPI` to close the NSL Manager, as shown in Listing 2-7.

Listing 2-7 Deinitializing the NSL Manager

```
NSLCloseNavigationAPI(myClientRef);
```

If this application is the last application that has a requirement for a particular plug-in, the NSL Manager unloads that plug-in from memory.

CHAPTER 3

Network Services Location Manager Reference

NSL Manager Functions

The NSL Manager functions are described in these sections:

- "Getting Information About the NSL Manager" (page 37)
- "Managing NSL Manager Sessions" (page 38)
- "Making a Lookup Request" (page 40)
- "Looking for Neighborhoods and Services" (page 44)
- "Managing Memory" (page 55)
- "Managing Services" (page 57)
- "NSL Manager Utility Functions" (page 59)
- "NSL Manager Application-Defined Routines" (page 78)

Getting Information About the NSL Manager

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

NSLLibraryPresent

Determines whether the NSL Manager is present.

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

DISCUSSION

The `NSLLibraryPresent` function returns `TRUE` when the NSL Manager is available.

NSLLibraryVersion

Determines which version of the NSL Manager is present.

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

DISCUSSION

The `NSLLibraryVersion` function returns the version of the NSL Manager installed on the system in hexadecimal format with the first set of two bytes representing the version number, the second set of two bytes representing the revision number, and the third set of two bytes representing the subrevision number.

Managing NSL Manager Sessions

NSLOpenNavigationAPI

Opens a session with the NSL Manager.

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

`newref`	On input, a pointer to an `NSLClientRef` in which the NSL Manager returns a value that your application uses in subsequent `NSLPrepareRequest` (page 3-42) and `NSLCloseNavigationAPI` calls (page 3-39).
function result	A value of `noErr` indicates that the session was opened and all available plug-ins loaded successfully. A value of `kNSLSomePluginsFailedToLoad` indicates that the session was opened and at least one plug-in loaded successfully. If `NSLOpenNavigationAPI` returns any of the following error codes, your application should not call any other NSL Manager functions: `kNSLNotInitialized`, `kNSLInsufficientSysVer`, `kNSLInsufficientOTVer`, `kNSLPluginLoadFailed`, or `kNSL68kContextNotSupported`.

DISCUSSION

The `NSLOpenNavigationAPI` function opens a session with the NSL Manager and returns an `NSLClientRef` that your application later uses to prepare NSL lookup requests and to close the NSL session. If no other application has opened a session, calling `NSLOpenNavigationAPI` initializes the NSL Manager. You must call `NSLOpenNavigationAPI` before you call any other NSL Manager functions.

The version of the NSL Manager that comes with the NSL SDK requires Mac OS version 9.0 or later and Open Transport 1.3 or later in order to initialize successfully.

NSLCloseNavigationAPI

Closes a session with the NSL Manager.

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

`theClient`	On input, the `NSLClientRef`, obtained by previously calling `NSLOpenNavigationAPI` (page 3-38), that identifies the session that is to be closed.

DISCUSSION

The `NSLCloseNavigationAPI` function closes the specified NSL Manager session.

▲ **WARNING**
If your application calls `NSLCloseNavigationAPI` while a lookup is in progress, any data that would have been returned is lost. ▲

Your application is responsible for reclaiming memory that it allocates for services lists, parameter blocks, and lookup requests. Your application should reclaim this memory by calling `NSLDisposeServicesList` (page 3-55), `NSLDeleteRequest` (page 3-56) and `NSLFreeTypedDataPtr` (page 3-61), respectively.

Making a Lookup Request

NSLMakeNewServicesList

Creates a services list.

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

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

function result A services list. `NSLMakeNewServicesList` returns `NULL` if it can't create the services list because, for example, there is not enough memory or because the NSL Manager is not initialized.

DISCUSSION

The `NSLMakeNewServicesList` function creates a services list and fills it with the names of the services specified in `initialServiceList`. After you create the services list, you can add the names of additional services by calling `NSLAddServiceToServicesList` (page 3-41).

When you have no further use for the services list, you can reclaim the memory allocated to it by calling `NSLDisposeServicesList` (page 3-55).

NSLAddServiceToServicesList

Adds the name of a service to a services list.

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

`serviceList`	On input, a services list previously created by calling `NSLMakeNewServicesList` (page 3-40).
`serviceType`	On input, a service type that is to be added to the services list.
function result	If the value of `NSLError.theErr` is `noErr`, the service was added to the list. Other possible values are `kNSLNotInitialized`, `kNSLBadServiceTypeErr`, `kNSLNullListPtr`, and `kNSLBadProtocolTypeErr`.

DISCUSSION

The `NSLAddServicesToServiceList` function adds the name of the specified service to a services list.

IMPORTANT
You must create `serviceList` by calling `NSLMakeNewServicesList` before you call `NSLAddServicesToServicesList`. ▲

Call `NSLAddServiceToServicesList` for each service that you want to add to the services list.

NSLPrepareRequest

Creates a lookup request.

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

`notifier`	On input, `NULL` (for synchronous lookups) or a value of type `NSLClientNotifyUPP` that points to your application’s notification routine (for asynchronous lookups). Your notification routine will be called when data is available, when the lookup is complete, or when an error occurs. If you don’t provide a notification routine, you should start searches from another thread so that your application can process events in a different thread.
`contextPtr`	On input, an untyped pointer to arbitratry data that the NSL 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 lookup request.
`theClient`	On input, an `NSLClientRef` obtained by previously calling `NSLOpenNavigationAPI` (page 3-38) that identifies the NSL Manager session.
`ref`	On output, a pointer to the resulting lookup request.
`bufPtr`	On input, a pointer to the buffer in which lookup results are to be placed.
`bufLen`	On input, the length of the buffer pointed to by `bufPtr`.
`infoPtr`	On output, `infoPtr` contains default information about how the search is to be conducted. Your application can change the defaults before it starts the lookup.

function result If the value of `NSLError.theErr` is `noErr`, the request was created. Other possible values include `kNSLNotInitialized`, `kNSLDuplicateSearchInProgress`, and `kNSLBadClientInfoPtr`.

DISCUSSION

The `NSLPrepareRequest` function creates a lookup request, which your application later uses as a parameter when it calls `NSLStartNeighborhoodLookup` (page 3-47) or `NSLStartServicesLookup` (page 3-49).

If `notifier` is null when you call `NSLPrepareRequest`, any lookup that uses the resulting lookup request is performed synchronously. `NSLStartServicesLookup` (page 3-49) and `NSLContinueLookup` (page 3-52) will return when the result buffer is full, the lookup is complete, or an error occurs. Your application can cause `NSLStartServicesLookup` and `NSLContinueLookup` to return 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` structure (page 3-90) pointed to by `infoPtr`.

Note
When performing synchronous searches, setting the `maxSearchTime` field may not cause the search to terminate within the specified time. Instead, set the `alertInterval` field to the desired maximum amount of search time and call `NSLCancelRequest` (page 3-55) to cancel the search when the alert interval expires. ◆

If `notifier` is a pointer to your application’s notification routine, your application’s notification routine will be called when the result buffer contains data, the result buffer is full, when the lookup 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` structure (page 3-90) pointed to by `infoPtr`.

The NSL Manager does not call your application’s notification routine at interrupt time, so your notification routine can allocate memory.

When your application no longer needs the lookup request, it should call `NSLDeleteRequest` (page 3-56) to reclaim memory associated with the request.

If `NSLPrepareRequest` returns `kDuplicateSearchInProgress`, there is an ongoing lookup that is using an identical `NSLRequestRef`. Your application can ignore this warning, delete the newly created `NSLRequestRef`, or cancel the lookup that is using the identical `NSLRequestRef`.

Looking for Neighborhoods and Services

NSLStandardGetURL

Displays a dialog box that allows the user to conduct a search.

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

`dialogOptions`	On input, a pointer to an `NSLDialogOptions` structure (page 3-93) whose fields specify the appearance of the dialog box. Call `NSLGetDefaultDialogOptions` to fill the fields of an `NSLDialogOptions` structure with the default dialog options. After calling `NSLGetDefaultDialogOptions`, you can customize the contents of the fields in the `NSLDialogOptions` structure.
`eventProc`	On input, a value of type `NSLEventUPP` that points to an application-defined system event callback routine as described in “System Event Callback Routine” (page 78) or `NULL`. If `eventProc` is `NULL`, your application will not receive update events while the “Select a Service” dialog box is displayed.
`eventcontextPtr`	On input, an untyped pointer to arbitrary data that the NSL Manager passes to the application-defined system event callback routine specified by `eventProc`. Your application can use

`contextPtr` to associate any particular execution of your system event callback routine with any particular call of the `NSLStandardGetURL` function.

`filterProc` On input, a value of type `NSLURLFilterUPP` that specifies your application-defined filter routine (page 3-78), or `NULL` if you do not have a filter routine. If specified, your filter routine will be called for each URL that is about to be displayed in the dialog box. If your filter routine returns `TRUE`, you have the option of specifying the name the URL is listed under by filling in the `displayString` parameter, which is passed as a parameter to your filter routine. If your filter routine returns `FALSE`, the URL is not displayed.

`serviceTypeList`

On input, a null-terminated string of tuples that describe the services that are to be searched for. The format of the tuples is as follows:

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, the value of `serviceTypeList` would be

```
"Web Servers, http,https;FTP Servers,ftp"
```

The result of setting `serviceTypeList` in this way would be a popup menu containing two items: “Web Servers” and “FTP Servers.” The result of a search performed on these two items would consist of a list HTTP and HTTPS services followed by a list of FTP services.

See the Discussion section below for information about using `serviceTypeList` to control the icon that is displayed for each service type.

`userSelectedURL`

On input, the address of a pointer to a string. On output, if `NSLStandardGetURL` returns `noErr`, `url` contains the null-terminated URL the user selected. When your application no longer needs `userSelectedURL`, it should call `NSLFreeURL` (page 3-62) to reclaim the memory associated with it.

function result	If `NSLStandardGetURL` returns `noErr`, the user selected a URL and it is stored in the `userSelectedURL` parameter. If `NSLStandardGetURL` returns `kNSLUserCanceled`, the user clicked the Cancel button in the dialog box and the `userSelectedURL` parameter is empty.

DISCUSSION

The `NSLStandardGetURL` 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 NSL Manager is responsible for displaying the neighborhoods, which it obtains by querying the NSL plug-ins that support the services specified by `serviceTypeList`.

The `NSLStandardGetURL` 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 use the `serviceTypeList` parameter to specify the display of an application-defined icon instead of an icon defined by the NSL Manager.

To cause the NSL Manager to display an application-defined icon, specify 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 NSL Manager’s unique icons will be displayed for HTTP and HTTPS services, the NSL Manager’s generic icon will be displayed for Telnet services, and the icon at resource ID 129 in your application’s resource fork will be displayed for NFS services.

IMPORTANT

Be sure to dispose of the `url` parameter by calling `NSLFreeURL`(page 62) when `urlSelectedParameter` is no longer needed. ▲

NSLStartNeighborhoodLookup

Looks for neighborhoods.

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

`ref`	On input, an `NSLRequestRef` created by previously calling `NSLPrepareRequest` (page 3-42).
`neighborhood`	On input, an `NSLNeighborhood` value created by previously calling `NSLMakeNewNeighborhood` (page 3-69). If `neighborhood` was created with a value of `name` that was `NULL`, `NSLStartNeighborhoodLookup` returns the first default neighborhood. If `neighborhood` was created with a value of `name` that is a name, `NSLStartNeighborhoodLookup` returns a related name. For example, if `neighborhood` was created with a value of `name` that is `apple.com`, `NSLStartNeighborhoodLookup` returns a subdomain of apple.com.
`asyncInfo`	On input, a pointer to the `asyncInfo` structure in whose `resultBuffer` field `NSLStartNeighborhood` is to store neighborhood lookup results.
function result	If the value of `NSLError.theErr` is `noErr`, `NSLStartNeighborhoodLookup` returned successfully. Possible errors are `kNSLNotInitialized`, `kNSLSearchAlreadyInProgress`, `kNSLNoPluginsForSearch`, `kNSLBufferTooSmallForData`, and `kNSLNullNeighborhoodPtr`.

DISCUSSION

The `NSLStartNeighborhoodLookup` function returns a neighborhood value that your application can use to define the scope of a subsequent service lookup.

IMPORTANT

For any `NSLRequestRef`, only one neighborhood or service lookup can be in progress at any one time. ▲

If `ref` was created with a value of `notifier` that is null, `NSLStartNeighborhoodLookup` operates synchronously. If `ref` was created with a value of `notifier` that is pointer to your application's notification routine, `NSLStartNeighborhoodLookup` operates asynchronously.

When `NSLStartNeighborhoodLookup` returns (if called synchronously) or when the NSL Manager calls your application's notification routine (if `NSLStartNeighborhoodLookup` is called asynchronously), your application should check the value of `asyncInfo.searchState`, which contains one of the following values:

```
kNSLSearchStateBufferFull = 1,
kNSLSearchStateOnGoing = 2,
kNSLSearchStateComplete = 3,
kNSLSearchStateStalled = 4,
kNSLWaitingForContinue = 5
```

If the value of `asyncInfo.searchState` is `kNSLSearchStatusBufferFull`, your application should process the data returned in `asyncInfo.resultBuffer`. Then it should call `NSLContinueLookup` (page 3-52) to resume the lookup.

IMPORTANT

Calling `NSLContinueLookup` will cause the information in the result buffer to be overwritten. ▲

If the value of `asyncInfo.searchState` is `kNSLSearchStateOnGoing`, the value of `asyncInfo.alertInterval` or `asyncInfo.alertThreshold` has been reached. Your application should process the data returned in `asyncInfo.resultBuffer`. Then it should call `NSLContinueLookup` to resume the lookup.

If the value of `asyncInfo.searchState` is `kNSLSearchStateComplete` and `NSLStartNeighborhoodLookup` does not return an error, the lookup is complete. Your application should process the data returned in `asyncInfo.resultBuffer`. If the value of `asyncInfo.searchState` is `kNSLSearchStateComplete` and `NSLStartNeighborhoodLookup` returns an error, the error is a fatal error and you cannot call `NSLContinueLookup`.

If the value of `asyncInfo.searchState` is `kNSLSearchStateStalled`, the value of `asyncInfo.alertInterval` or `asyncInfo.maxSearchTime` has been reached, but there is no data in the result buffer. One or more plug-ins for this lookup is waiting to receive data from a server but has not yet timed out. If the value of `asyncInfo.searchState` is `noErr`, your application should call `NSLContinueLookup` to resume the lookup.

If `NSLStartNeighborhoodLookup` returns `kNSLBufferTooSmallForData`, the value of `asyncInfo.maxBuffserSize` is too small to hold an item that would otherwise have been returned. Your application can cancel and restart the lookup, or it can call `NSLContinueLookup` to resume the lookup even though some data will be lost.

IMPORTANT

If more than one plug-in participates in a lookup, the result buffer may contain valid data even though `NSLStartNeighborhoodLookup` returns an error code from one of the plug-ins. If the value of `asyncInfo.searchState` is `kNSLSearchStateOngoing`, the error code is not fatal. Your application should process the data in the result buffer and can call `NSLContinueLookup` to continue the lookup. ▲

SEE ALSO

`NSLGetNextNeighborhood` (page 3-64) for information about processing the data in the result buffer.

NSLStartServicesLookup

Looks for services.

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

`ref`	On input, an `NSLRequestRef` created by previously calling `NSLPrepareRequest` (page 3-42).
`neighborhood`	On input, an `NSLNeighborhood` value created by previously calling `NSLMakeNewNeighborhood` (page 3-69).
`requestData`	On input, a parameter block that describes the search parameters. To format `requestData` properly, call `NSLMakeServicesRequestPB` (page 3-70).

`asyncInfo`	On input, a pointer to a `NSLClientAsyncInfo` structure (page 3-90) obtained by calling `NSLPrepareRequest`.
function result	If the value of `NSLError.theErr` is `noErr`, `NSLStartServicesLookup` returned successfully. Other possible values are `kNSLNotInitialized`, `kNSLSearchAlreadyInProgress`, `kNSLNoPluginsForSearch`, `kNSLNullNeighborhoodPtr`, and `kNSLBufferTooSmallForData`.

DISCUSSION

The `NSLStartServicesLookup` function starts a service lookup.

IMPORTANT

For any `NSLRequestRef`, only one neighborhood or service lookup can be ongoing at any one time. ▲

If `ref` was created with a value of `notifier` that is null, `NSLStartServicesLookup` operates synchronously. If `ref` was created with a value for `notifier` that is pointer to your application's notification routine, `NSLStartServicesLookup` operates asynchronously.

▲ **WARNING**

In addition to calling `WaitNextEvent` or `SystemTask` from your main event loop, your application must call `YieldToAnyThread`. If your application does not call `YieldToAnyThread`, service lookups will fail and your application will appear to hang. ▲

When `NSLStartServicesLookup` returns (if called synchronously) or when the NSL Manager calls your application's notification routine (if `NSLStartServicesLookup` is called asynchronously), your application should check the value of `asyncInfo.searchState`, which contains one of the following values:

```
kNSLSearchStateBufferFull = 1,
kNSLSearchStateOnGoing = 2,
kNSLSearchStateComplete = 3,
kNSLSearchStateStalled = 4
```

If the value of `asyncInfo.searchState` is `kNSLSearchStatusBufferFull`, your application should process the data returned in `asyncInfo.resultBuffer`. Then it should call `NSLContinueLookup` (page 3-52) to resume the lookup.

IMPORTANT

Calling `NSLContinueLookup` will cause the information in the result buffer to be overwritten. ▲

If the value of `asyncInfo.searchState` is `kNSLSearchStateOnGoing`, the value of `asyncInfo.alertInterval` or `asyncInfo.alertThreshold` has been reached. Your application should process the data returned in `asyncInfo.resultBuffer`. Then it should call `NSLContinueLookup` to resume the lookup.

If the value of `asyncInfo.searchState` is `kNSLSearchStateComplete`, the lookup is complete.Your application should process the data returned in `asyncInfo.resultBuffer`.

If the value of `asyncInfo.searchState` is `kNSLSearchStateStalled`, the value of `asyncInfo.alertInterval` or `asyncInfo.maxSearchTime` has been reached, but there is no data in the result buffer. One or more plug-ins for this lookup is waiting to receive data from a server but has not yet timed out. If the value of `asyncInfo.searchState` is `noErr`, your application should call `NSLContinueLookup` to resume the lookup.

If `NSLStartServicesLookup` returns `kNSLBufferTooSmallForData`, the value of `asyncInfo.maxBuffserSize` is too small to hold an item that would otherwise have been returned. Your application can cancel and restart the lookup, or it can call `NSLContinueLookup` to resume the lookup even though some data will be lost.

IMPORTANT

If more than one plug-in participates in a lookup, the result buffer may contain valid data even though `NSLStartServicesLookup` returns an error code from one of the plug-ins. If the value of `asyncInfo.searchState` is `kNSLSearchStateBufferFull`, your application should process the data in the result buffer. ▲

To cancel an ongoing lookup, call `NSLCancelRequest` (page 3-55).

SEE ALSO

`NSLGetNextUrl` (page 3-65) for information about processing the data in the result buffer. `NSLDeleteRequest` (page 3-56) for information about deleting a lookup request that is no longer needed.

NSLContinueLookup

Continues a lookup.

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

`asyncInfo`	A pointer to the `NSLClientAsyncInfo` structure (page 3-90) for this lookup.
function result	If the value of `NSLError.theErr` is `noErr`, `NSLContinueLookup` returned successfully. Possible errors include `kNSLNotInitialized`, `kNSLNoContextAvailable`, `kNSLBadClientInfoPtr`, and `kNSLCannotContinueLookup`, and `kNSLBufferTooSmallForData`.

DISCUSSION

The `NSLContinueLookup` function continues a service lookup or a neighborhood lookup that has paused because `NSLStartNeighborhoodLookup`, `NSLStartServicesLookup`, or a previous call to `NSLContinueLookup` has returned, or because your application's notification routine has been called. Your application should check the value of `asyncInfo.searchState`, which contains one of the following values:

```
kNSLSearchStateBufferFull = 1,
kNSLSearchStateOnGoing = 2,
kNSLSearchStateComplete = 3,
kNSLSearchStateStalled = 4
```

If the value of `asyncInfo.searchState` is `kNSLSearchStatusBufferFull`, your application should process the data returned in `asyncInfo.resultBuffer`. Then it should call `NSLContinueLookup` again to resume the lookup.

IMPORTANT

Calling `NSLContinueLookup` will cause the information in the result buffer to be overwritten. ▲

If the value of `asyncInfo.searchState` is `kNSLSearchStateOnGoing`, the value of `asyncInfo.alertInterval` or `asyncInfo.alertThreshold` has been reached. Your application should process the data returned in `asyncInfo.resultBuffer`. Then it should call `NSLContinueLookup` again to resume the lookup.

If the value of `asyncInfo.searchState` is `kNSLSearchStateComplete`, the lookup is complete.Your application should process the data returned in `asyncInfo.resultBuffer`.

If the value of `asyncInfo.searchState` is `kNSLSearchStateStalled`, the value of `asyncInfo.alertInterval` or `asyncInfo.maxSearchTime` has been reached, but there is no data in the result buffer. One or more plug-ins for this lookup is waiting to receive data from a server but has not yet timed out. If the value of `asyncInfo.searchState` is `noErr`, your application should call `NSLContinueLookup` again to resume the lookup.

If `NSLContinueLookup` returns `kNSLBufferTooSmallForData`, the value of `asyncInfo.maxBuffserSize` is too small to hold an item that would otherwise have been returned. Your application can cancel and restart the lookup, or it can call `NSLContinueLookup` again to resume the lookup even though some data will be lost.

IMPORTANT

If more than one plug-in participates in a lookup, the result buffer may contain valid data even though `NSLContinueLookup` returns an error code from one of the plug-ins. If the value of `asyncInfo.searchState` is `kNSLSearchStateBufferFull`, your application should process the data in the result buffer. ▲

To cancel an ongoing lookup, call `NSLCancelRequest` (page 3-55).

SEE ALSO

`NSLGetNextUrl` (page 3-65) for information about processing the data in the result buffer when looking for services. `NSLDeleteRequest` (page 3-56) for information about deleting a lookup request that is no longer needed.

NSLErrorToString

Obtains information about an error.

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

`theErr`	On input, an `NSLError` structure (page 3-94) whose `theErr` field contains an NSL error number.
`errorString`	On input, a pointer to the buffer in which `NSLErrorToString` 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 `NSLErrorToString` 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 `NSLErrorToString` returned successfully. If `NSLError.theContext` is zero and `NSLError.theErr` contains an error number that is not within the range of NSL error numbers, `NSLErrorToString` returns `kNSLBadReferenceErr`.

DISCUSSION

The `NSLErrorToString` function obtains information about an `NSLError` structure (page 3-94) so that your application can display an appropriate error message. The `NSLError` structure may have been returned by the NSL Manager or by an NSL plug-in. For any given lookup, search results may be returned by more than one plug-in. You may not want to display an error message if one or more plug-ins return data without error.

NSLCancelRequest

Cancels an ongoing lookup.

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

`ref`	On input, the `NSLRequestRef` obtained by previously calling `NSLPrepareRequest` (page 3-42) for the lookup that is to be canceled.
function result	If the value of `NSLError.theErr` is `noErr`, the request was canceled successfully. Other possible values are `kNSLNotInitialized` and `kNSLBadReferenceErr`.

DISCUSSION

The `NSLCancelRequest` function cancels an ongoing lookup. Any outstanding I/O is also canceled.

Managing Memory

NSLDisposeServicesList

Disposes of a services list.

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

`theList`	On input, the services list that is to be disposed of.

DISCUSSION

The `NSLDisposeServicesList` function reclaims memory by disposing of a services list. Once you've incorporated the information in a services list into a request parameter block, you can dispose of the services list.

Calling `NSLCloseNavigationAPI` (page 3-39) does not reclaim memory allocated for services lists, so your application should dispose of services lists before it closes the NSL session.

NSLDeleteRequest

Deletes a lookup request.

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

`ref`	On input, the `NSLRequestRef` obtained by previously calling `NSLPrepareRequest` (page 3-42) for the lookup request that is to be deleted.
function result	If the value of `NSLError.theErr` is `noErr`, the lookup request was deleted. Other possible values are `kNSLNotInitialized` and `kNSLBadReferenceErr`.

DISCUSSION

The `NSLDeleteRequest` function deletes the specified lookup request and deallocates memory associated with it, including the `NSLClientAsyncInfo` structure. If a lookup is in progress for the specified lookup request when you call `NSLDeleteRequest`, the lookup is terminated and any outstanding I/O is lost.

`The NSLDeleteRequest` function does not deallocate memory associated with the services list or request parameter blocks. To deallocate memory for services lists, call `NSLDisposeServicesList` (page 3-55); to deallocate memory for parameter blocks, call `NSLFreeTypedDataPtr` (page 3-61).

Managing Services

NSLStandardRegisterURL

Registers the URL of a service.

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

`urlToRegister`	On input, a value of type `NSLPath` specifying the URL to register.
`neighborhoodToRegisterIn`	On input, a value of type `NSLNeighborhood` specifying the neighborhood in which to register the service, or `NULL`. If `NULL`, the plug-ins that handle the service specified by `urlToRegister` determine the neighborhood in which the service is registered.
function result	If the value returned by `NSLStandardRegisterURL` is `noErr`, the service was registered. The `NSLStandardRegisterURL` function returns `kNSLNoSupportForService`, which indicates that none of the currently installed plug-ins support the service for which registration is requested or that none of the currently installed plug-ins support any type of service registration.

DISCUSSION

The `NSLStandardRegisterURL` function registers the specified URL with the NSL Manager without requiring that your application previously call `NSLOpenNavigationAPI`.

Note

`NSLStandardRegisterURL` returns `kNSLBadURLSyntax` if `urlToRegister` contains illegal characters. If portions of the URL that you are registering contain illegal characters, call `NSLHexEncodeText` (page 3-68) to encode the illegal characters before you call `NSLStandardRegisterURL`. ◆

An application that provides a network service should call `NSLStandardRegisterURL` as part of its standard startup procedure.

▲ **WARNING**
In addition to calling `WaitNextEvent` or `SystemTask` from your main event loop, your application must call `YieldToAnyThread`. If your application does not call `YieldToAnyThread`, services will not be registered and your application will appear to hang. ▲

Your application should deregister the service by calling `NSLStandardDeregisterURL` as part of its standard shutdown routine to indicate that the service is no longer available.

Note
The `NSLStandardRegisterURL` function is available in NSL 1.1 and later, and supersedes the `NSLRegisterService` function, which was provided in NSL 1.0. ◆

NSLStandardDeregisterURL

Deregisters a service registered by `NSLStandardRegisterURL`.

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

`urlToDeregister`
On input, a value of type `NSLPath` specifying the URL that is to be deregistered.

`neighborhoodToDeRegisterIn`
On input, a value of type `NSLNeighborhood` specifying the neighborhood in which to deregister the service, or `NULL`. If `NULL`, the plug-ins that handle the service specified by `urlToRegister` determine the neighborhood from which the service is deregistered.

function result If the value returned by `NSLStandardDeregisterURL` is `noErr`, the service was deregistered. The `NSLStandardDeregisterURL` function returns `kNSLNoSupportForService`, which indicates that none of the currently installed plug-ins support the service for which deregistration is requested or don't support registration at all. Other possible errors include `kNSLNotInitialized`.

DISCUSSION

The `NSLStandardDeregisterURL` function deregisters the service specified by `urlToDeregister`. You should call `NSLStandardDeregisterURL` as part of your standard shutdown procedure for services that your application registered by calling `NSLStandardRegisterURL`.

Note
`NSLStandardDeregisterURL` returns `kNSLBadURLSyntax` if `urlToDeregister` contains illegal characters. If portions of the URL that you are deregistering contain illegal characters, call `NSLHexEncodeText` (page 3-68) to encode the illegal characters before you call `NSLStandardDeregisterURL`. ◆

Note
The `NSLStandardDeregisterURL` function is available in NSL 1.1 and later, and supersedes the `NSLRegisterService` function, which was provided in NSL 1.0. ◆

NSL Manager Utility Functions

You can use these utility functions to manipulate create, manipulate, and dispose of neighborhoods, to create and dispose of request parameter blocks, to encode and decode characters in a URL, and to manipulate service lists.

- `NSLCopyNeighborhood` (page 3-60) makes a copy of a neighborhood.
- `NSLFreeNeighborhood` (page 3-61) disposes of a neighborhood value.
- `NSLFreeTypedDataPtr` (page 3-61) deallocates the memory associated with a request parameter block.

- `NSLFreeURL` (page 3-61) deallocates the memory associated with a URL.
- `NSLGetDefaultDialogOptions` (page 3-62) sets the fields of an `NSLDialogOptions` structure to the default values.
- `NSLHexDecodeText` (page 3-68) decodes the portion of a URL that has been encoded.
- `NSLHexEncodeText` (page 3-68) encodes a portion of a URL.
- `NSLGetNameFromNeighborhood` (page 3-63) obtains the name of a neighborhood.
- `NSLGetNeighborhoodLength` (page 3-64)obtains the length of a neighborhood.
- `NSLGetNextNeighborhood` (page 3-64) obtains a pointer to the name of the next neighborhood in a buffer.
- `NSLGetNextUrl` (page 3-65) obtains a pointer to the next URL in a buffer.
- `NSLGetServiceFromURL` (page 3-66) obtains the service portion of a URL.
- `NSLMakeNewNeighborhood` (page 3-69) creates a neighborhood value.
- `NSLMakeServicesRequestPB` (page 3-70) creates a service request parameter block.
- `NSLServiceIsInServiceList`(page 3-71) determines whether a service is in a service list.

NSLCopyNeighborhood

Copies a neighborhood.

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

`neighborhood`	On input, a value of type `NSLNeighborhood` representing the neighborhood that is to be copied.
function result	An `NSLNeighborhood` value that can be used in a subsequent call to `NSLStartServicesLookup`. If `NSLCopyNeighborhood` can’t create the copy of `neighborhood`, it returns `NULL`. This might happen, for example, if there is not enough memory.

DISCUSSION

The `NSLCopyNeighborhood` function creates a copy of the specified neighborhood. When an application calls the NSL Manager's `NSLStartNeighborhoodLookup` function, it passes a neighborhood as a parameter. The NSL Manager passes the neighborhood to one or more plug-ins. The calling application can delete the neighborhood at any time, so upon receipt of a neighborhood, each plug-in should call `NSLCopyNeighborhood` to make a copy of it.

When you have no further use for an `NSLNeighborhood` value, you can reclaim the memory allocated to it by calling `NSLFreeNeighborhood` (page 3-61).

NSLFreeNeighborhood

Disposes of an `NSLNeighborhood` value.

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

`neighborhood`	On input, the `NSLNeighborhood` value that is to be disposed of.
function result	An `NSLNeighborhood` whose value is always `NULL`.

DISCUSSION

The `NSLFreeNeighborhood` function disposes of an `NSLNeighborhood` value and reclaims that memory that was allocated to it.

NSLFreeTypedDataPtr

Frees memory allocated for a request parameter block.

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

`nslTypeData`	On input, a value of type `NSLTypedDataPtr` obtained by previously calling `NSLMakeServicesRequestPB` (page 3-70).
function result	A value of type `NSLTypedDataPtr` whose value is always `NULL`.

DISCUSSION

The `NSLFreeTypedDataPtr` function frees memory that your application caused to be allocated when it previously called `NSLMakeServicesRequestPB` (page 3-70). Your application should call `NSLFreeTypedDataPtr` (page 3-61) when it has no further use for the parameter block specified by `nslTypeData`.

NSLFreeURL

Frees memory allocated for a URL.

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

`URL`	On input, a pointer to a character string obtained by previously calling `NSLStandardGetURL` (page 3-44).
function result	`NULL`, which allows `URL` to be set to `NULL` and the memory associated with `URL` to be freed in one step.

DISCUSSION

The `NSLFreeURL` function frees the memory that is allocated for `URL` when an application calls `NSLStandardGetURL` (page 3-44). Your application should call `NSLFreeURL` when it has no further use for the URL that the user selects in the “Select a Service” dialog box.

NSLGetDefaultDialogOptions

Assigns the default dialog options to an `NSLDialogOptions` structure.

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

`dialogOptions`	On input, a pointer to a `dialogOptions` structure. On output, the fields of the `dialogOptions` structure are set with the default values.
function result	A result code. See “NSL Manager Result Codes” (page 101) for possible values.

DISCUSSION

The `NSLGetDefaultDialogOptions` function sets the fields of an `NSLDialogOptions` structure to values that are the defaults for the “Select a Service” dialog box. For the default values, see the section “NSLDialogOptions” (page 93).

NSLGetNameFromNeighborhood

Locates the neighborhood name in a neighborhood.

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

`neighborhood`	On input, a value of type `NSLNeighborhood` from which the name is to be obtained.
`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

The `NSLGetNameFromNeighborhood` function locates the name in a neighborhood so that your application can, for example, display the name. The name that `NSLGetNameFromNeighborhood` locates is not null-terminated. Use the value pointed to by `length` to determine the length of the name.

Note
The `NSLGetNameFromNeighborhood` function does not allocate any memory. ◆

NSLGetNeighborhoodLength

Obtains the length of a neighborhood.

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

`neighborhood`	On input, a value of type `NSLNeighborhood` whose length is to be obtained.
function result	The length in bytes of the specified neighborhood.

DISCUSSION

The `NSLGetNeighborhoodLength` function obtains the length of the specified neighborhood.

NSLGetNextNeighborhood

Obtains the next neighborhood in a buffer.

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

`infoPtr`	On input, a pointer to an `NSLClientAsyncInfo` structure (page 3-90) whose `resultBuffer` field may contain another neighborhood.
`neighborhood`	On input, a pointer to a value of type `NSLNeighborhood`. 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 `resultBuffer`. A value of `FALSE` indicates that there are no more neighborhoods in `resultBuffer`.

DISCUSSION

The `NSLGetNextNeighborhood` function obtains the starting position and the length of the next neighborhood in a result buffer.

If you want to keep a copy of the neighborhood, call `NSLCopyNeighborhood` (page 3-60) to copy the neighborhood from the buffer.

If you want to get the name of the neighborhood, call `NSLGetNameFromNeighborhood` (page 3-63).

NSLGetNextUrl

Obtains information about the next URL in a buffer.

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

`infoPtr`	On input, a pointer to an `NSLClientAsyncInfo` structure (page 3-90) whose `resultBuffer` field may contain a URL.
`urlPtr`	On output, if a URL was found in `resultBuffer`, 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 `resultBuffer`. A value of `FALSE` indicates that there are no more URLs in `resultBuffer`.

DISCUSSION

The `NSLGetNextUrl` function obtains the starting position and the length of the next URL in a result buffer. You call `NSLGetNextUrl` to parse the URLs returned by a previous call to `NSLStartServicesLookup`.

NSLGetServiceFromURL

Obtains the service portion of a URL.

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

`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` contains the length in bytes of `svcString`.
function result	A value of `noErr` indicates that `NSLGetServiceFromURL` successfully obtained the service portion of the specified URL.

DISCUSSION

The `NSLGetServiceFromURL` function obtains the service portion of a URL.

NSLHexDecodeText

Decodes the encoded portion of a URL.

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

`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 `encodedText`.
`decodedTextBuffer`	On input, a pointer to a buffer in which the decoded text is to be stored. On output, `decodedTextBuffer` contains the decoded text.
`decodedTextBufferLen`	On input, a pointer to a value of type `UInt16` containing the maximum length of `decodedTextBuffer`. On output, `decodedTextBufferLen` contains 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 the contents of the string pointed to by `encodedText` does not match the content of the string pointed to by `decodedTextBuffer`.
function result	A value of `noErr` indicates that `NSLHexDecodeText` returned successfully.

DISCUSSION

The `NSLHexDecodeText` function decodes the portion of a URL that has been encoded by calling `NSLHexEncodeText` (page 3-68).

If NSLHexDecodeText returns noErr, and textChanged is FALSE, decodedTextBuffer points to a copy of the data pointed to by encodedText. That is, the data pointed to by encodedText is copied regardless of whether any decoding takes place.

NSLHexEncodeText

Encodes a portion of a URL.

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

rawText	On input, a pointer to a character array containing the portion of the 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 may contain illegal characters is Kevs/G3. In this example, the forward slash (/) is an illegal character.
rawTextLen	On input, a value of type UInt16 containing the length in bytes of rawText.
newTextBuffer	On input, a pointer to a buffer. On output, the buffer contains the encoded text.
newTextBufferLen	On input, a pointer to an unsigned short. On output, newTextBufferLen contains the length of the encoded text pointed to by newTextBuffer.
textChanged	On input, a pointer to a Boolean value. On output, textChanged is TRUE if the encoded data is different from the original data and FALSE if the encoded data has not changed.
function result	A value of noErr indicates that NSLHexEncodeText returned successfully.

DISCUSSION

The `NSLHexEncodeText` function uses the US ASCII character code set to encode the portion of a URL that may contain illegal characters. Illegal characters are hexadecimal values in ranging from 01 to 1F, from 80 to FF, and 7F, as well as the following characters:

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

In addition, `NSLHexEncodeText` also encodes the following characters that are reserved for URL syntax:

```
; / ? : @ = % &
```

Call `NSLHexDecodeText` (page 3-67) to decode a string that has been encoded.

Note
The `NSLStandardRegisterURL` function returns an error if the URL that you try to register contains illegal characters. ◆

If `NSLHexEncodeText` returns `noErr` and `textChanged` is `FALSE`, `newTextBuffer` points to a copy of the data pointed to by `rawText`. That is, the data pointed to by `rawText` is copied regardless of whether any encoding takes place.

NSLMakeNewNeighborhood

Creates a neighborhood.

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

`name`	On input, a pointer to a null-terminated string containing a name. If `dns` is specified in `protocolList`, the value of `name` should be a domain name, such as `apple.com`. If `slp` is specified in `protocolList`, the value of name should be a scope. Other types of names may be appropriate depending on the installed plug-ins. To create an `NSLNeighborhood` that can be used to obtain a list of default neighborhoods when you call `NSLStartNeighborhoodLookup`, set `name` to `NULL`.

`protocolList`	On input, a pointer to a comma-separated, null-terminated list of protocols (such as `dns,slp`) that are to participate in a lookup conducted with the resulting `NSLNeighborhood` value. If the value of `protocolList` is `NULL`, all available protocols will participate in the lookup.
function result	An `NSLNeighborhood` value that can be used in a subsequent call to `NSLStartServicesLookup`. If `NSLMakeNewNeighborhood` can't create an `NSLNeighborhood` value, it returns `NULL`. This might happen, for example, if there is not enough memory.

DISCUSSION

The `NSLMakeNewNeighborhood` function creates an `NSLNeighborhood` value that defines the boundary of a subsequent search.

When you have no further use for an `NSLNeighborhood` value, you can reclaim the memory allocated to it by calling `NSLFreeNeighborhood` (page 3-61).

NSLMakeServicesRequestPB

Creates a request parameter block.

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

`serviceList`	On input, an `NSLServicesList` created by previously calling `NSLMakeNewServicesList`(page 40).
`newDataPtr`	On input, the address of the `NSLTypedDataPtr` at which `NSLMakeServicesRequestPB` is to place the resulting parameter block.
function result	A value of `noErr` indicates that `NSLMakeServicesRequestPB` returned successfully. Possible errors include `kNSLBadDomainErr`.

DISCUSSION

The `NSLMakeServicesRequestPB` function creates a parameter block that is formatted properly for use with subsequent calls to `NSLStartServicesLookup`(page 49).

NSLServiceIsInServiceList

Determines whether a service is in a service list.

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

`serviceList`	On input, an `NSLServicesList` created by previously calling `NSLMakeNewServicesList` (page 3-40).
`svcToFind`	On input, a value of type `NSLServiceType` that contains a null-terminated service that is to be checked.
function result	A value of `TRUE` indicates that the service specified by `svcToFind` is in the service list specified by `serviceList`; otherwise, `NSLServiceIsInServiceList` returns `FALSE`.

DISCUSSION

The `NSLServiceIsInServiceList` function determines whether the service specified by `svcToFind` is in the service list specified by `serviceList`.

NSL Manager Plug-in Utility Functions

NSL plug-ins use the utility functions described in this section to obtain error strings, to manage threads, and to parse parameter blocks.

- `NSLGetErrorStringsFromResource` (page 3-72) gets the error and solution string from a plug-in's `'NSLE'` resource fork using the ID for the plug-in's error resource.

- `NSLDisposeThread` (page 3-73) disposes of a thread created by calling `NSLNewThread`.
- `NSLNewThread` (page 3-74) creates a thread for use by an NSL plug-in.
- `NSLParseServiceRegistrationPB` (page 3-76) parses a service registration parameter block.
- `NSLParseServicesRequestPB` (page 3-77) parses a services request parameter block.

NSLGetErrorStringsFromResource

Obtain error strings from the plug-in's resource fork.

```
OStatus NSLGetErrorStringsFromResource (
                    OStatus theErr,
                    FSSpecPtr fileSpecPtr,
                    SInt16 errorResID,
                    char* errorString,
                    char* solutionString);
```

`theErr`	On input, a value of type `OSStatus` containing an error code.
`fileSpecPtr`	On input, a value of type `FSSpecPtr` that points to the plug-in's resource that contain a list of error codes, error strings, and solution strings.
`errorResID`	On input, a value of type `SInt16` containing the resource ID of the plug-in's `'NSLE'` resource.
`errorString`	On input, a pointer to a character string that is at least 256 bytes in length. On output, `errorString` contains the error string that corresponds to the error code specified by `theErr`.
`solutionString`	On input, a pointer to a character string that is at least 256 bytes in length. On output, `solutionString` contains the solution string that corresponds to the error code specified by `theErr`.

DISCUSSION

The `NSLGetErrorStringsFromResource` function reads the specified resource ID in the resource fork specified by `fileSpecPtr` looking for an error code that matches `theErr`. If `GetNSLErrorStringFromResource` finds a match, it stores the error and solution strings associated with that error code in `errorString` and `solutionString`, respectively.

The sample plug-in in the NSL SDK includes a template for creating an `'NSLE'` resource.

NSLDisposeThread

Disposes of a thread created by calling `NSLNewThread`.

```
OSErr NSLDisposeThread (ThreadID threadToDump,
                        void * threadResult,
                        Boolean recycleThread);
```

`threadToDump`	On input, a value of type `ThreadID` that represents the thread you want to dispose of.
`threadResult`	On input, a pointer to a arbitrary data.
`recycleThread`	On input, a Boolean value that is `TRUE` if you want to return the thread to the pool of premade threads or `FALSE` if you want to dispose of the thread entirely.
function result	A value of `noErr` indicates that thread was disposed of. Possible errors include `threadNotFoundErr` (the specified thread does not exist), and `threadProtocolErr` (the `threadToDump` parameter specifies the application thread).

DISCUSSION

The `NSLDisposeThread` function disposes of a thread that was created by calling `NSLNewThread` (page 3-74).

Note
Returning from a thread causes the thread to be disposed of automatically. ◆

NSLNewThread

Creates a thread for use by an NSL plug-in.

```
OSErr NSLNewThread (ThreadStyle threadStyle,
                    ThreadEntryProcPtr threadEntry,
                    void * threadParam,
                    Size stackSize,
                    ThreadOptions options,
                    void** threadResult,
                    ThreadID * threadMade);
```

`threadStyle`	On input, a value of type `ThreadStyle`. Set `threadStyle` to `kCooperativeThread`.
`threadEntry`	On input, a value of type `ThreadEntryProcPtr` that points to the function that is to be executed when the created thread is resumed.
`threadParam`	On input, a pointer to an arbitrary application-defined value that is passed to `threadEntry` for its use.
`stackSize`	On input, a value of type Size that specifies the requested stack size of the new thread. The stack size must be large enough to handle saved thread context, normal application stack usage, interrupt handling routine, and CPU exceptions. By setting `stackSize` to zero, `NSLNewThread` uses the default stack size for cooperative threads.
`options`	On input, a value of type `ThreadOptions` that specifies optional characteristics of the created thread. The options are summed together to create the desired combination. See the Discussion section for information on available options.
`threadResult`	On input, a pointer to a pointer to an arbitrary value. When a called thread terminates, its result is stored in `threadResult`. If you are not interested in the result, set `threadResult` to `NULL`.
`threadMade`	On input, a pointer to value of type `ThreadID`. On output, `threadMade` contains the thread's identifier.

function result	A value of `noErr` indicates that the thread was created. Possible errors include `memFullErr` (not enough memory to create the thread), `threadTooManyReqsErr` (no thread are available), and `paramErr` (for example, an unknown thread style was specified).

DISCUSSION

The `NSLNewThread` function creates or allocates a thread for use by an NSL plug-in and makes the thread's identifier available in the `threadMade` parameter. NSL plug-ins that would otherwise use the Thread Manager to create or allocate threads must call `NSLNewThread` instead of the Thread Manager's `NewThread` function. To dispose of a thread created by calling `NSLNewThread`, NSL plug-ins must call `NSLDisposeThread`. NSL plug-ins can call any other Thread Manager function to manage threads created by `NSLNewThread`.

The following constants are available for use with the `options` parameter:

`kNewSuspend`	Allocates the new thread so that it begins in the `kStoppedThreadState` and is ineligible for execution. Otherwise, the thread is created in the `kReadyThreadState` and is eligible to run.
`kUsePremadeThread`	Allocates the new thread from the pool of premade threads. By default, threads allocated from the thread pool are made on a stack size best-fit basis.
`kExactMatchThread`	Requires that the stack size of threads allocated from the pool of premade threads exactly match the stack size specified by the `stackSize` parameter.
`kCreateIfNeeded`	Allocates an entirely new thread if the pool of premade threads is exhausted.

NSLParseServiceRegistrationPB

Parse a registration parameter block.

```
OSStatus NSLParseServiceRegistrationPB (NSLTypedDataPtr newDataPtr,
                        NSLNeighborhood* neighborhoodPtr,
                        UInt16* neighborhoodLen,
                        char** urlPtr,
                        UInt16* urlLen);
```

`newDataPtr`	On input, a value of type `NSLTypedDataPtr` that points to the registration parameter block that is to be parsed.
`neighborhoodPtr`	On input, a pointer of type `NSLNeighborhood`. On output, `neighborhoodPtr` pointer to the neighborhood portion of the `newDataPtr` parameter.
`neighborhoodLen`	On input, a pointer to a value of type `UInt16`. On output, `neighborhoodLen` points to a value that contains the length in bytes of the neighborhood pointed to by `neighborhoodPtr`.
`urlPtr`	On input, the address of a pointer to a character string. On output, `urlPtr` contains the address of a pointer that points to the portion of the `newDataPtr` parameter containing the URL that is to be registered.
`urlLen`	On input, a pointer to a value of type `UInt16`. On output, `urlLen` points to a value that contains the length of the URL referenced by `urlPtr`.
function result	A value of `noErr` indicates that `NSLParseServicesRegistrationPB` returned successfully. The `NSLParseServicesRegistrationPB` function returns `kBadDataTypeError` if `newDataPtr` is not a valid request parameter block.

DISCUSSION

The `NSLParseServiceRegistrationPB` function parses a registration parameter block. When the NSL Manager calls an NSL plug-in's `Register` routine (page 3-81), the NSL Manager passes the registration parameter block to the plug-in. The plug-in calls `NSLParseServiceRegistrationPB` to determine the location in the registration parameter block of the neighborhood and the URL of the service that is to be registered.

NSLParseServicesRequestPB

Parse a services request parameter block.

```
OSStatus NSLParseServicesRequestPB (NSLTypedDataPtr newDataPtr,
                      char** serviceListPtr,
                      UInt16* serviceListLen);
```

`newDataPtr`	On input, a value of type `NSLTypedDataPtr` which points to the address of the request parameter block that is to be parsed.
`serviceListPtr`	On input, the address of a pointer to a character string. On output, `serviceListPtr` is set to the address of a pointer that points to the portion of `NewDataPtr` that holds the list of services for which a search is to be conducted.
`serviceListLen`	On input, a pointer to a value of type `UInt16`. On output, `serviceListLen` points to a value that contains the length in bytes of the list of services referenced by `serviceListPtr`.
function result	A value of `noErr` indicates that `NSLParseServicesRequestPB` returned successfully. The `NSLParseServicesRequestPB` function returns `kBadDataTypeError` if `newDataPtr` is not a valid request parameter block.

DISCUSSION

The `NSLParseServicesRequestPB` function parses a request parameter block. When the NSL Manager calls an NSL plug-in's `StartServicesLookup` routine (page 3-83), the NSL Manager passes the parameter block to the plug-in, which calls `NSLParseServicesRequestPB` to determine the location of the service list and attributes that the application has specified for the lookup.

NSL Manager Application-Defined Routines

Filter Callback Routine

When you call `NSLStandardGetURL` (page 3-44) to display the "Select a Service" dialog box that allows the user to specify the services to search for and the neighborhoods in which to search, you can provide a callback routine that filters the search results. This is how you would declare your filter callback routine if you were to name it `MyNSLFilterProcPtr`:

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

`url`	A pointer to a string containing the URL to filter.
`displayString`	A value of type `Str255`. If your filter callback routine returns `TRUE`, it could set `displayString` to a value that you want to be displayed in the directory listing part of the "Select a Service" dialog box. By default, any data in a URL that follows the tag "?NAME=" is for display purposes only and is dropped from the url that is returned to the calling application.
result	Your filter routine should return `TRUE` if the URL specified by `url` should be displayed. It should return `FALSE` to prevent the URL from being displayed.

DISCUSSION

Only those results that pass through your filter callback routine are displayed in the Directory list area of the "Select a Service" dialog box.

System Event Callback Routine

If your application calls `NSLStandardGetURL` (page 3-44), you may want to provide a routine that handles system events that may occur while the "Select a

Service" dialog box is active. A typical system event callback routine might be defined in the following way:

```
typedef pascal void (NSLEventProcPtr) (
                    EventRecord* newevent,
                    void* userContext);
```

`newevent`	A pointer to a structure of type `eventRecord` that describes the event that triggered the callback. For more information on the `EventRecord` structure, see *Inside Macintosh: Overview.*
`userContext`	An untyped pointer to arbitrary data that your application previously passed to `NSLStandardGetURL` (page 3-44).
result	Your system event callback routine should process the system event.

DISCUSSION

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

NSL Manager Plug-in Routines

NSL plug-ins reside in the Extensions folder inside the System Folder. The icon for NSL plug-ins is shown in Figure 3-1.

Figure 3-1 NSL plug-in icon

The creator code for an NSL plug-in is `'NSLp'` and the type code is `'shlb'`.

Every NSL plug-in must provide the following routines for the NSL Manager to call:

- `InitPlugin`(page 80). Initializes the plug-in.
- `Register`(page 81). Registers a service that the plug-in supports.
- `StartNeighborhoodLookup`(page 82). Initiates a search for neighborhoods.
- `StartServicesLookup`(page 83). Initiates a search for services.
- `ContinueLookup`(page 85). Continues a search for neighborhoods or services.
- `ErrNumToString`(page 87). Provides strings that contain a description and a resolution for the specified error number.
- `CancelLookup`(page 88). Cancels a search for services or neighborhoods.
- `Deregister`(page 88). Deregisters a service that was previously registered.
- `KillPlugin`(page 89). Prepares the plug-in for being unloaded from memory.

In addition to providing these routines, every NSL plug-in must have an NSLI resource that provides information about the plug-in to the NSL Manager. See `SamplePlugin.rsrc` in the sample code that comes with the NSL Developer's Kit.

Note
Instead of calling the Thread Manager to create and dispose of threads, NSL plug-ins must call `NSLNewThread`(page 74) and `NSLDisposeThread`(page 73), respectively. ◆

InitPlugin

Initializes the plug-in.

```
OSStatus InitPlugin (void);
```

result	A value of `noErr` indicates that `InitPlugin` successfully initialized the plug-in.

DISCUSSION

The `InitPlugin` routine allocates memory for the plug-in and opens network connections that the plug-in will use.

Register

Registers services.

```
OSStatus Register (NSLTypedDataPtr dataPtr);
```

`dataPtr`	A value of type `NSLTypedDataPtr` that points to a value that specifies the services that are to be registered.
result	A value of `noErr` indicates that `Register` successfully registered the specified services. To indicate that the services were not successfully registered, `Register` can return any NSL error code. For example, if `Register` cannot parse `dataPtr`, it should return `kNSLBadDataTypeErr`.

DISCUSSION

The `Register` routine registers the services specified by `dataPtr`. The NSL Manager calls a plug-in's `Register` routine in response to an application that calls the NSL Manager function `NSLStandardRegisterURL`(page 57).

▲ **WARNING**
The calling application can delete `dataPtr` at any time, so the `Register` routine should make a copy of it as soon as possible. ▲

To parse the services specified by `dataPtr`, the `Register` routine calls `NSLParseServiceRegistrationPB`(page 76).

Note
If your plug-in doesn't set the `SupportsRegistration` flag in the NSLI resource, the plug-in's `Register` routine will not be called. ◆

StartNeighborhoodLookup

Looks for neighborhoods.

```
OSStatus StartNeighborhoodLookup (NSLNeighborhood neighborhood,
                        NSLMgrNotifyUPP notifier,
                        NSLPluginAsyncInfoPtr pluginInfo);
```

`neighborhood`	A value of type `NSLNeighborhood` that identifies the neighborhood in which the lookup is to be conducted.
`notifier`	A value of type `NSLMgrNotifyUPP` that points to the NSL Manager notification routine. The `StartNeighborhoodLookup` routine should call the notification routine when the result buffer contains one item, the lookup is complete, or an error has occurred. The NSL Manager's notification routine allocates memory, so you should not call it at interrupt time.
`pluginInfo`	A value of type `NSLPluginAsyncInfoPtr` that points to an `NSLPluginAsyncInfo`(page 99) structure whose `clientRef` and `requestRef` fields identify the application and request associated with the lookup that is to be started, whose `maxSearchTime` field may limit the amount of time that is to be spent on the lookup, whose `resultBuffer` field is to be changed to point to the plug-in's lookup result, and whose `searchState` and `searchResult` fields are used to store status information about the lookup.
result	A value of `noErr` indicates that `StartNeighborhoodLookup` completed successfully. The `StartNeighborhoodLookup` routine can return any NSL error code to indicate that it did not start the lookup. A value of `kNSLBadDataTypeErr` indicates that the search was not started because one or more of the input parameters is invalid.

DISCUSSION

The `StartNeighborhoodLookup` routine performs the lookup specified by `neighborhood`. The NSL Manager calls a plug-in's `StartNeighborhoodLookup` routine in response to an application that calls the NSL Manager function `NSLStartNeighborhoodLookup(page 47)`.

To obtain the name of the neighborhood specified by `neighborhood`, the `StartNeighborhoodLookup` routine calls `NSLGetNameFromNeighborhood(page 63)`.

▲ **WARNING**
The calling application can delete `neighborhood` at any time, so the `StartNeighborhoodLookup` routine should call `NSLCopyNeighborhood`(page 60) to copy the neighborhood as soon as possible. ▲

A plug-in's `StartNeighborhoodLookup` routine stores one neighborhood in its result buffer, changes `pluginInfo.resultBuffer` to point to its result buffer, sets `pluginInfo.bufferLen` to the length of the valid data in its result buffer, and calls the NSL Manager's notification routine.

If the value of `pluginInfo.maxSearchTime` is a non-zero positive value when the plug-in's `StartNeighborhoodLookup` routine is called, `StartNeighborhoodLookup` routine should maintain a count of the time in ticks that it spends on this lookup.

To maintain context information about this lookup, the `StartNeighborhoodLookup` routine can use `pluginInfo.pluginContext`.

▲ **WARNING**
The NSL Manager's notification routine allocates memory, so you should not call it at interrupt time. ▲

StartServicesLookup

Looks for services.

```
OSStatus StartServicesLookup (NSLNeighborhood neighborhood,
                         NSLTypedDataPtr dataPtr,
                         NSLMgrNotifyUPP notifier,
                         NSLPluginAsyncInfo* pluginInfo);
```

`neighborhood`	A vale of type `NSLNeighborhood` that specifies the neighborhood in which the service lookup is to be conducted.
`dataPtr`	A value of type `NSLTypedDataPtr` that points to a value that identifies the parameters for a lookup.

`notifier`	A value of type `NSLMgrNotifyProcUPP` that points to the NSL Manager notification routine. The `StartNeighborhoodLookup` routine should call the notification routine when the result buffer contains one item, the lookup is complete, or an error has occurred. The NSL Manager's notification routine allocates memory, so you should not call it at interrupt time.
`pluginInfo`	A value of type `NSLPluginAsyncInfoPtr` that points to an `NSLPluginAsyncInfo`(page 99) structure whose `clientRef` and `requestRef` fields identify the application and request associated with the lookup that is to be started, whose `maxSearchTime` field may specify a limit on the amount of time that is to be spent on the search, whose `resultBuffer` field should be changed to point to the plug-in result, and whose `searchState` and `searchResult` fields are used to store status information about the lookup.
result	A value of `noErr` indicates that `StartServicesLookup` completed successfully. The `StartServicesLookup` routine can return any NSL error code to indicate that it did not start the lookup. A value of `kNSLBadDataTypeErr` indicates that the search was not started because one or more of the input parameters is invalid.

DISCUSSION

The `StartServicesLookup` routine performs the lookup specified by `dataPtr`. The NSL Manager calls a plug-in's `StartServicesLookup` routine in response to an application that calls the NSL Manager's `NSLStartServicesLookup` function (page 3-49).

IMPORTANT
The `StartServicesLookup` routine may be called synchronously or asynchronously, so it should be prepared to handle both modes. ▲

To obtain the name of the neighborhood specified by `neighborhood`, the `StartNeighborhoodLookup` routine calls `NSLGetNameFromNeighborhood` (page 3-63).

▲ **WARNING**
The calling application can delete `neighborhood` and `dataPtr` at any time, so the plug-in should copy these values as soon as it receives them. To copy `neighborhood`, call `NSLCopyNeighborhood` (page 3-60). ▲

To parse the lookup parameters specified by `dataPtr`, the `StartServicesLookup` routine calls `NSLParseServicesRequestPB` (page 3-77).

Upon receipt of a URL, a plug-in's `StartServicesLookup` routine places the URL in its result buffer, changes `pluginInfo.resultBuffer` to point to its result buffer, sets `pluginInfo.bufferLen` to the length of the valid data in its result buffer, and calls the NSL Manager's notification routine.

If the value of `pluginInfo.maxSearchTime` is a non-zero positive value, `StartServicesLookup` should limit the overall time for this lookup to the specified time in ticks if the specified time is less than the plug-in's own limit.

To maintain context information about this lookup, the `StartServicesLookup` routine can use `pluginInfo.pluginContextPtr`.

▲ **WARNING**
The NSL Manager's notification routine allocates memory, so you should not call it at interrupt time. ▲

ContinueLookup

Continues a lookup.

```
OSStatus ContinueLookup (NSLMgrNotifyUPP notifier,
                         NSLPluginAsyncInfo* pluginInfo);
```

`notifier`	A value of type `NSLMgrNotifyProcPtr` that points to the NSL Manager notification routine. The `ContinueLookup` routine should call the notification routine when the result buffer contains one item, the lookup is complete, the maximum search time has been reached, or an error has occurred. The NSL Manager's notification routine allocates memory, so you should not call it at interrupt time.
`pluginInfo`	A value of type `NSLPluginAsyncInfoPtr` that points to an `NSLPluginAsyncInfo`(page 99) structure whose `clientRef` and `requestRef` fields identify the application and request associated with the lookup that is to be continued, whose `maxSearchTime` field may specify a limit on the amount of time that is to be spent on the search, whose `resultBuffer` field should be changed to point to the plug-in result, and whose `searchState`

and `searchResult` fields are used to store status information about the lookup. Your `ContinueLookup` routine should make a copy of `pluginInfo` because the client application may free `pluginInfo` at any time.

result	A value indicating that `ContinueLookup` completed successfully. The `ContinueLookup` routine can return any NSL error code to indicate that it did not continue the lookup.

DISCUSSION

The `ContinueLookup` routine continues a lookup that is in progress. The lookup that is to be continued is identified by `pluginInfo.requestRef`. The NSL Manager calls a plug-in's `ContinueLookup` routine in response to an application that calls the NSL Manager function `NSLContinueLookup`(page 52).

Upon receipt of an item, a plug-in's `ContinueLookup` routine places the item in its result buffer, changes `pluginInfo.resultBuffer` to point to its result buffer, sets `pluginInfo.bufferLen` to the length of the valid data in its result buffer, and calls the NSL Manager's notification routine.

If the value of `pluginInfo.maxSearchTime` was a non-zero positive value when the plug-in's `StartServicesLookup` routine was called, `ContinueLookup` routine should maintain a count of the time that it has spent on this lookup and limit the search to the specified time.

Note
For a particular lookup, the value of `pluginInfo.maxSearchTime` should not change between calls to `StartServicesLookup` and `ContinueLookup` or between successive calls to `ContinueLookup`. ◆

▲ **WARNING**
The NSL Manager's notification routine allocates memory, so you should not call it at interrupt time. ▲

ErrNumToString

Provides strings that correspond to error codes.

```
OSStatus ErrNumToString (
                        OSStatus errNum,
                        char* errString,
                        char* theSolution);
```

`errNum`	The error code, previously returned by one of the plug-in's routines, for which a string description and string solution are to be obtained.
`errString`	A pointer to the character string in which `ErrorToString` is to place a null-terminated string of up to 256 bytes that describes the error condition that corresponds to `theError`.
`theSolution`	A pointer to the character string in which `ErrNumToString` is to place a null-terminated string of up to 256 bytes that suggests a solution to the error condition that corresponds to `errNum`.
result	A value indicating that `ErrNumToString` completed successfully. The `ErrNumToString` routine can return any NSL error code to indicate that it did not provide the requested strings.

DISCUSSION

The `ErrNumToString` routine stores in `errString` a string that describes the error number specified by `errNum` and stores in `theSolution` a suggested solution. The NSL Manager calls a plug-in's `ErrNumToString` routine when an application calls the NSL Manager function `NSLErrorToString` `NSLErrorToString`(page 54).

When `ErrNumToString` returns, the NSL Manager returns the strings to the application. The application may choose to display `errString` and `theSolution` to the user, so both strings should be suitable for display.

If your plug-in has a resource of type`'NSLE'`, it can call `NSLGetErrorStringsFromResource(page 72)` to obtain the information required to fill in `errString` and `theSolution`. For information about the `'NSLE'` resource, see the section “`NSL Error Resource” (page 101)`.

IMPORTANT

A plug-in's `ErrNumToString` routine should be able to provide a value for `errString` and `theSolution` for every error code that the plug-in returns. ▲

CancelLookup

Cancels a lookup.

```
OSStatus CancelLookup (NSLPluginAsyncInfoPtr pluginInfo);
```

`infoPtr`	A value of type `NSLPluginAsyncInfoPtr` that points to the `NSLPluginAsyncInfo`(page 99) structure whose `requestRef` field identifies the lookup that is to be canceled.
result	A value indicating that `CancelLookup` successfully canceled the lookup. The `CancelLookup` routine can return any NSL error code to indicate that it did not cancel the lookup.

DISCUSSION

The `CancelLookup` routine cancels the lookup associated with `pluginInfo`. The NSL Manager calls a plug-in's `CancelLookup` routine when an application calls the NSL Manager's `NSLCancelRequest` function (page 3-55).

Deregister

Deregisters services.

```
OSStatus Deregister (NSLTypedDataPtr dataPtr);
```

`dataPtr`	A value of type `NSLTypedDataPtr` that identifies the services that are to be deregistered.

result	A value of `noErr` indicates that `Deregister` successfully deregistered the specified services. To indicate that the services were not successfully deregistered, `Deregister` can return any NSL error code. For example, if `Deregister` cannot parse `dataPtr`, it should return `kNSLBadDataTypeErr`.

DISCUSSION

The `Deregister` routine deregisters the services specified in `dataPtr`. The NSL Manager calls a plug-in's `Deregister` routine in response to an application that calls the NSL Manager function `NSLStandardDeregisterURL`(page 58).

▲ **WARNING**
The calling application can delete `dataPtr` at any time, so the `Deregister` routine should make a copy of it as soon as possible. ▲

To parse the services specified by `dataPtr`, the `Deregister` routine calls `NSLParseServiceRegistrationPB`(page 76).

KillPlugin

Prepares the plug-in for unloading.

```
OSStatus KillPlugin(Boolean forceQuit);
```

`forceQuit`	A Boolean value. If `forceQuit` is `TRUE`, the `KillPlugin` routine must deinitialize itself completely. If `forceQuit` is `FALSE`, the `KillPlugin` routine can conduct all or part of its deinitialization procedures at its discretion.
result	A value of `noErr` indicates that `KillPlugin` successfully deinitialized the plug-in. If the value of `forceQuit` is `FALSE`, `KillPlugin` can return any NSL error code to indicate that it needs to remain in memory.

DISCUSSION

The `KillPlugin` routine prepares the plug-in to be unloaded from memory by stopping any lookups that the plug-in is conducting, closing open network connections, and deallocating memory that the plug-in has allocated.

The NSL Manager calls a plug-in's `KillPlugin` routine in response to an application that calls the function `NSLCloseNavigationAPI`(page 39) when that application is the last application that has an open NSL Manager session.

If the plug-in needs to remain in memory (for example, to handle requests for registered services) and if `forceQuit` is `FALSE`, the plug-in can return an error code and remain in memory. However, if `forceQuit` is `TRUE`, the plug-in must deinitialize itself completely and prepare to be unloaded from memory.

NSL Manager Structures

The following structures supply the information that applications need to call NSL Manager functions.

- The structure `NSLClientAsyncInfo`(page 90) contains all of the information required to start a neighborhood or service lookup.
- The structure `NSLDialogOptions`(page 93) contains all of the information required to call `NSLStandardGetURL` to display the "Select a Service" dialog box.
- The structure `NSLError`(page 94) is used by NSL Manager functions to return an error code as well as contextual information about that error code.

The structure `NSLPluginAsyncInfo`(page 99) is used by plug-ins to receive information from the NSL Manager about a lookup request.

NSLClientAsyncInfo

The `NSLClientAsyncInfo` structure contains information about how a neighborhood or a service lookup is to be conducted and where lookup results are to be stored. You obtain a pointer to a `NSLClientAsyncInfo` structure by calling `NSLPrepareRequest`(page 42), and you pass that pointer as a parameter when you call `NSLStartNeighborhoodLookup(page 47)`, `NSLStartServicesLookup`(page 49), or `NSLContinueLookup`(page 52).

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

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

```
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;
```

Field descriptions

`clientContextPtr`	A value set by the application for its own use.
`mgrContextPtr`	A value set by the NSL Manager for its own use.
`resultBuffer`	A pointer to the buffer that contains lookup results.
`bufferLen`	The number of bytes in `resultBuffer` that contain valid data.
`maxBufferSize`	The length of `resultBuffer`.

`startTime`	Used by the NSL Manager for internal purposes. Your application should not modify this field.
`intStartTime`	Used by the NSL 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 a plug-in may impose.
`alertInterval`	An application-specified value that defines in ticks the interval at which the application's notification routine is to be called or the interval at which `NSLStartServicesLookup`, `NSLStartNeighborhoodLookup`, or `NSLContinueLookup` are to return. 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 or `NSLStartServicesLookup`, `NSLStartNeighborhoodLookup`, or `NSLContinueLookup` to return whenever the specified number of items have been placed in `resultBuffer`. Typically, applications that cause `NSLStartServicesLookup` or `NSLStartNeighborhoodLookup` to operate asynchronously set `alertThreshold` to 1, and applications that cause `NSLStartNeighborhood` or `NSLStartServicesLookup` to operate synchronously set `alertThreshold` to zero, which indicates that no alert threshold is specified. The default value is zero.

`searchState`	A value that describes the current search state. The value can be one of the following:
	
	`kNSLSearchStateBufferFull= 1,`
	`kNSLSearchStateOnGoing = 2,`
	`kNSLSearchStateComplete = 3,`
	`kNSLSearchStateStalled = 4`

`searchResult`	An `NSLError` structure containing an error code that the NSL Manager or a plug-in may have returned.
`searchDataType`	An event code that indicates whether the information stored in this `NSLClientAsyncInfo` structure pertains to a

neighborhood lookup (kNSLNeighborhoodLookupDataEvent) or a service lookup (kNSLServicesLookupDataEvent).

NSLDialogOptions

The NSLDialogOptions structure contains information that controls the appearance of the "Select a Service" dialog box, which is displayed by calling NSLStandardGetURL.

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

Field descriptions

version	A value that specifies the version.
dialogOptionFlags	A bit map whose value controls whether to display the URL text field that allows the user to enter a URL.
windowTitle	A string containing the name that is to appear in the title bar. The default title is "Select a Service".
actionButtonLabel	A string containing the name that is to be used as the label for the action button. If NULL, the default name is used. The default is "Choose."
cancelButtonLabel	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 is "Cancel."
message	A string containing the text that is to be used for a custom prompt. If NULL, no prompt is displayed.

The dialogOptions flags enumeration defines constants for use in dialogOptionsFlags field:

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

NSLError

The `NSLError` structure is used by certain NSL Manager functions to return an error code as well as contextual information about that error code.

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

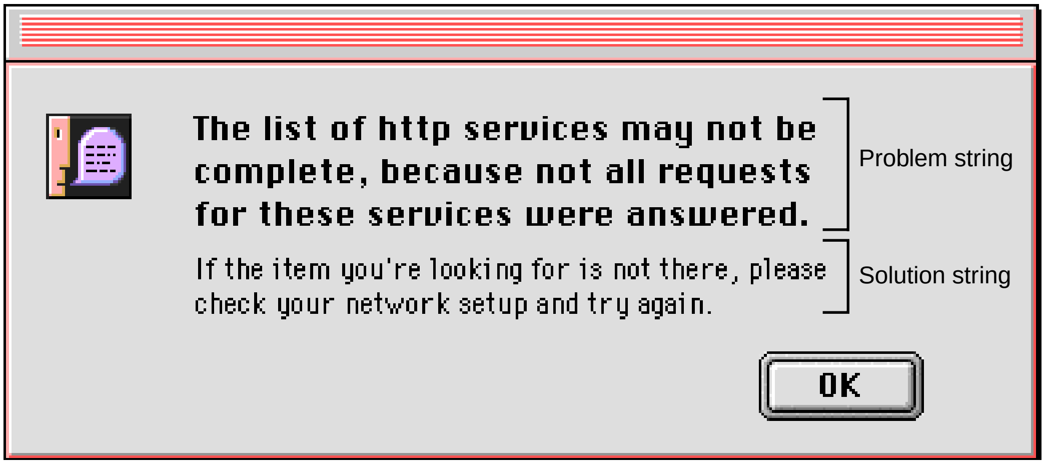
Field descriptions

`theErr`	The error code.
`theContext`	A value used by the NSL Manager to determine whether it generated the error code or whether a plug-in generated error code. If a plug-in generated the error code, the value of `theContext` allows the NSL Manager to identify the responsible plug-in.

Comparing the constant `kNSLErrorNoErr` to the value returned by an function that returns an `NSLError` structure is a simple way to determine whether an error occurred.

If you want to display information about the error to the user, your application should call `NSLErrorToString`(page 54) to obtain two strings — a problem string and a solution string. To display the strings, use a movable modal dialog box, as shown in Figure 3-2.

Figure 3-2 Standard alert dialog box


The list of http services may not be complete, because not all requests for these services were answered.

If the item you're looking for is not there, please check your network setup and try again.

Table 3-1 lists the problem and solution strings for error conditions that commonly occur.

Table 3-1 NSL problem and solution strings

Error string	Solution String	Source
The request could not be completed due to an internal error in NSL.	Please try again. If you still have problems, try restarting the application or the computer, then try your request again.	NSL Manager
The NSL Manager could not be initialized.	Restart the computer and try again.	NSL Manager
The installed system software does not support this version of NSL.	Install Mac OS System Software version 8.7 or later.	NSL Manager
Your computer is not connected to the network.	Check your network settings and make sure all networking cables are properly attached. Then try your request again.	NSL Manager
Unable to load NSL plugins during startup.	Restart your computer and try your request again.	NSL Manager
The NSL Manager could not find any plugins to load.	Make sure you have at least one NSL Plugin in your Extensions folder.	NSL Manager
The NSL Manager could not load any plugins to handle your request.	Check your network settings and make sure all networking cables are properly attached. Then try your request again.	NSL Manager
Some, but not all, NSL plugins failed to load.	You can continue, since at least one plugin is available. Look in the Extensions Manager control panel to see which plugin is available.	NSL Manager
The NSL UI Library is not available.	Please make sure this shared library is in your Extensions folder and try again.	NSL Manager

continued

Error string	Solution String	Source
A file needed for this operation was not available.	Make sure you have installed all components correctly.	NSL Manager
The last command could not be completed because there is not enough memory.	Quit some applications and close some windows to make more memory available.	NSL Manager, DNS plug-in, LDAP plug-in, NBP plug-in, SLP plug-in
The last command could not be completed because your hard disk is full.	Quit the application and delete some items to free up some disk space.	DNS plug-in, LDAP plug-in, SLP plug-in
The last command could not be completed because the startup disk is locked.	Unlock your startup disk and try again.	DNS plug-in, SLP plug-in
The correct version of Open Transport is not available.	Install Open Transport 1.3 or later	NSL Manager, DNS plug-in, LDAP plug-in, SLP plug-in
The <*name-of-plug-in*> plugin could not complete the request due to a network error.	Check to make sure that your network is set up correctly.	DNS plug-in, SLP plug-in
The <*name-of-plug-in*> plugin could not complete the request due to an internal error.	Please try again. If you still have problems, try restarting the application or the computer, then try the request again.	DNS plug-in, LDAP plug-in, NBP plug-in, SLP plug-in
Unable to reach Name server.	Open your TCP/IP control panel and enter a valid Name server address.	DNS plug-in
The name server is not responding.	This is probably due to access restrictions at the name server. Contact your network administrator for help with name server access.	DNS plug-in

continued

Error string	Solution String	Source
The LDAP plugin could not complete the request because no LDAP server is available.	Make sure that your LDAP server is working and that the server name is specified in the Internet control panel. To see LDAP information, set the Internet control panel to Advanced user mode.	LDAP plug-in
The LDAP plugin could not complete the request because the LDAP server did not respond.	Make sure that your LDAP server is working and that the correct server name is specified in the Internet control panel. To see LDAP information, set the Internet control panel to Advanced user mode.	LDAP plug-in
The LDAP plugin could not complete the request because it could not find a nameserver.	Make sure you have specified a valid DNS nameserver in the TCP/IP control panel.	LDAP plug-in
The LDAP plugin could not complete the request because LDAP returned nil data.	Please try again. If it still fails, try to make your request more specific and try the request again.	LDAP plug-in
Invalid hostname for LDAP server.	Make sure that your LDAP server is working and that the server name is specified in the Internet control panel. To see LDAP information, set the Internet control panel to Advanced user mode.	LDAP plug-in
The LDAP plugin could not complete the request because a nil pointer was encountered.	This problem is usually related to memory. Quit some applications and try the request again.	LDAP plug-in

continued

Error string	Solution String	Source
The LDAP plugin could not complete the request because a buffer is not big enough.	This problem is usually related to memory. Quit some applications and try the request again.	LDAP plug-in
The LDAP plugin could not complete the request because it encountered a bad parameter.	Please try again. If you still have problems, try restarting the application or the computer, then try the request again.	LDAP plug-in
The SLP Plugin could not be used due to an error with the preferences file.	Please make sure that the SLP Preferences file is valid or remove it so that the SLP Plugin can create a new default preferences file.	SLP plug-in

NSLPluginAsyncInfo

The NSL Manager passes an `NSLPluginAsyncInfo` structure as a parameter to the plug-in’s `StartNeighborhoodLookup`(page 82), `StartServicesLookup`(page 83), and `ContinueLookup`(page 85) routines. The `NSLPluginAsyncInfo` structure contains all of the information that the plug-in needs to start or continue a lookup. The plug-in uses the `NSLPluginAsyncInfo` structure to maintain state information about an ongoing lookup request and to return information about the lookup to the NSL Manager.

```
struct NSLPluginAsyncInfo
{
    void *          mgrContextPtr;
    void *          pluginContextPtr;
    void *          pluginPtr;
    char *          resultBuffer;
    long            bufferLen;
    long            maxBufferSize;
    UInt32          maxSearchTime;
    UInt32          reserved1;
    UInt32          reserved2;
    UInt32          reserved3;
    NSLClientRef    clientRef
    NSLRequestRef   requestRef
    NSLSearchState  searchState;
    OSStatus        searchResult;
};
typedef struct NSLPluginAsyncInfo NSLPluginAsyncInfo;
typedef NSLPluginAsyncInfo * NSLPluginAsyncInfoPtr;
```

Field descriptions

`mgrContextPtr`	A value set by the NSL Manager for its own use.
`pluginContextPtr`	A value set by the plug-in for its own use.
`pluginPtr`	A pointer to the plug-in object that is waiting for the application to call `NSLContinueLookup`(page 52).
`resultBuffer`	A pointer to the buffer that the plug-in can use to store lookup results.
`bufferLen`	The length of valid data in `resultBuffer`.
`maxBufferSize`	The maximum length of `resultBuffer`.
`maxSearchTime`	The maximum length of time to search in ticks. A value of zero indicates an unlimited search time.
`Reserved1`	Reserved.
`Reserved2`	Reserved.
`Reserved3`	Reserved.
`clientRef`	A value identifying the application that made the request.
`requestRef`	A value specifying the lookup request.
`searchState`	A value that the plug-in sets to indicate the current state of the lookup. The value can be one of the following: `kNSLSearchStateBufferFull= 1,` `kNSLSearchStateOnGoing = 2,` `kNSLSearchStateComplete = 3,` `kNSLSearchStateStalled = 4`
`searchResult`	An `NSLError` structure that the plug-in uses to return error information.

NSL Error Resource

If your plug-in has an 'NSLE' resource containing error codes and their corresponding error strings and error solution strings, your plug-in's ErrNumToString routine can call the NSL Manager utility function NSLGetErrorStringsFromResource to obtain the error string and error solution string for a given error code.

The NSL SDK includes a template for creating your plug-in's 'NSLE' resource.

NSL Manager Result Codes

All of the NSL Manager functions return a result code. The result codes specific to the NSL Manager are listed here. In addition, NSL Manager functions may return other Mac OS result codes, which are described in *Inside Macintosh*.

noErr	0	No error.
kNSLNotInitialized	-4199	The NSL Manager could not be initialized.
kNSLInsufficientSysVer	-4198	The installed version of the Mac OS does not support the NSL Manager. (For the NSL Manager SDK, Version 9.0 or later is required.)
kNSLInsufficientOTVer	-4197	The installed version of Open Transport does support the NSL Manager. (Open Transport 1.3 or later is required.)
kNSLNoElementsInList	-4196	A specified list is empty.
kNSLBadReferenceErr	-4195	The specified NSLClientRef or NSLRequestRef is invalid.
kNSLBadServiceTypeErr	-4194	The specified service type is not supported.

`kNSLBadDataTypeErr`	-4193	The specified parameter is not of the correct data type.
`kNSLBadNetConnection`	-4192	A network error occurred. AppleTalk or TCP/IP may be turned off, or the computer may not be connected to the network.
`kNSLNoSupportForService`	-4191	No plug-in supports the requested service registration or deregistration.
`kNSLInvalidPluginSpec`	-4190	The `theContext` field of the specified `NSLError` structure is invalid.
`kNSLRequestBufferAlreadyInList`	-4189	Reserved.
`kNSLNoContextAvailable`	-4188	The `asyncInfo` parameter provided in a call to `NSLContinueLookup` is invalid.
`kNSLBufferTooSmallForData`	-4187	The application's result buffer is too small to store the data returned by a plug-in.
`kNSLCannotContinueLookup`	-4186	The lookup cannot be continued due to an error condition or a bad state.
`kNSLBadClientInfoPtr`	-4185	The specified `NSLClientAsyncInfoPtr` is invalid.
`kNSLNullListPtr`	-4184	The pointer to the specified list is invalid.
`kNSLBadProtocolTypeErr`	-4183	The specified `NSLServiceType` is empty.
`kNSLPluginLoadFailed`	-4182	During system initialization, the NSL Manager was unable to load one of the plug-ins.
`kNSLNoPluginsFound`	-4181	During system initialization, the NSL Manager was unable to find any valid plug-ins to load.

`kNSLSearchAlreadyInProgress`	-4180	A search is already in progress for the specified `clientRef`.
`kNSLNoPluginsForSearch`	-4179	None of the installed plug-ins are able to respond to the lookup request.
`kNSLNullNeighborhoodPtr`	-4178	The pointer to a neighborhood is invalid.
`kNSLSomePluginsFailedToLoad`	-4177	During system initialization, the NSL Manager was unable to load some plug-ins.
`kNSLErrNullPtrError`	-4176	A specified pointer is invalid.
`kNSLUILibraryNotAvailable`	-4174	The NSL UI Library is not available.
`kNSLBadURLSyntax`	–4172	The caller passed a URL that contains illegal characters.
`kNSLSchedulerError`	–4171	A custom thread routine encountered an error.
`kNSL68kContextNotSupported`	–4170	NSL was called from a non-PPC application.

Index