DNS Service Discovery Programming Guide

Networking > Bonjour

2005-11-09

Apple Inc.
© 2005 Apple Computer, Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

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

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

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

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

Apple, the Apple logo, Bonjour, iChat, iTunes, Mac, and Mac OS are trademarks of Apple Inc., registered in the United States and other countries.

Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Simultaneously published in the United States and Canada.

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

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

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

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

Contents

2005-11-09 | © 2005 Apple Computer, Inc. All Rights Reserved.

Introduction to DNS Service Discovery

This document describes how to use the DNS Service Discovery API in your program. The DNS Service Discovery API helps you to perform three main tasks:

- Registering a service
- Browsing for services
- Resolving service names to host names

In support of these main tasks, this API can directly assist you in performing two subsidiary tasks:

- Enumerating domains (finding recommended service domains)
- Updating registrations (changing your DNS registration data dynamically)

The DNS Service Discovery API is part of Bonjour, Apple's implementation of zero-configuration networking (ZEROCONF). For an overview of how Bonjour works, please read *Bonjour Overview* prior to using DNS Service Discovery.

Note: This document describes the socket-based DNS Service Discovery API, which is the recommended API that's available in Mac OS X v10.3 and later. The alternative Mach-based API is deprecated.

Who Should Read This Document?

This document is meant for developers who want to use Bonjour in their applications. There are a number of reasons to use the DNS Service Discovery API over the Network Services APIs (NSNetServices and CFNetServices):

- You are writing BSD-style applications that will not need to link to higher level frameworks.
- You are writing cross-platform programs (DNS Service Discovery API is available on Mac OS X, Windows, and other POSIX compatible operating systems).
- You are writing an application that requires special purpose low-level routines, such as registering individual records or being able to use only specific network interfaces.

If none of those features are necessary for your program, it is highly recommended that you take a look at NSNetService and CFNetService first.

Organization of This Document

This document contains the following articles:

- "Registering and Terminating a Service" (page 9) explains how to register and terminate your service with the mDNSResponder daemon.
- "Browsing for Network Services" (page 13) describes how to browse for services.
- "Resolving the Current Address of a Service" (page 17) explains how to get information on a service based on its name, registration type, and domain.
- "Enumerating Domains" (page 21) helps you understand how to find domains recommended for registration and browsing.
- "Using DNS Service Discovery in Windows" (page 23) gives an overview for how to implement DNS Service Discovery in a Windows-based application.

Before You Start

The next few paragraphs describe some things you should know about this API before attempting any of the tasks.

Most functions in this API do not return all of their data using their function return or parameter block. Instead, they require you to provide a callback function that can handle data sent asynchronously.

Your callback function may be called multiple times in response to a single function call on your part. For example, you might request a list of available services. Your callback would be called once for each available service that matches your request, then called again whenever a matching service is added or removed.

Some functions return error codes in the usual way, but many do not. In these cases, any error code and status flags are sent to your callback function as part of the asynchronous reply, along with—or instead of—any returned data.

Most of the functions in this API use a common set of parameters to describe services. You will need to supply some or all of these parameters, depending on the purpose of your call. In many cases, you will provide some parameters, such as the domain and type of service, and your callback function will receive data corresponding to other parameters, such as the service name of a matching service.

Here is a list of the common parameters required by the DNS Service Discovery API:

- Name—human readable name of the service, such as `Sales Laser Printer`.
- Registration type—the service type followed by the protocol name and separated by a dot (`.`); `_printer._tcp` is an example.
- Domain—the domain for the service, typically `NULL`.

- Full domain name—the name that uniquely identifies a service. A full domain name is the concatenation of the name, registration type, and domain. Because the dot (`.`) character is used as a separator, any dot characters in the name portion of a full domain name must be escaped by a backslash character (`\`). If the name contains a literal backslash, the backslash must also be escaped by a backslash character. Here is an example of a full domain name: `Dr\.Smith's Home\\Office Server._http._tcp.local`.
- Port—the port number for the service in network byte order.
- Text record—an optional record containing any additional information that may be needed to use the service, such as a print queue name.

Requirements

The DNS Service Discovery API requires the services of the mDNSResponder daemon. Mac OS X, versions 10.2 and later, include an mDNSResponder daemon as part of the operating system. Apple also provides source code for an mDNSResponder daemon to Darwin developers as part of versions 10.2 and later. This API is also available in Bonjour for Windows and when mDNSResponder source code is downloaded and incorporated in the Linux, Solaris, and FreeBSD operating systems.

Note: Apple encourages hardware developers to embed the Darwin mDNSResponder daemon code in their hardware.

Limitations

The DNS Service Discovery API does not perform network access setup for services. Similarly, the DNS Service Discovery API does not provide a network connection from an application to a service. It allows applications to browse for services, or to ask for them by name, and provides the IP address, port, and so on. You can use BSD sockets to connect to a service over the network.

For More Information

For additional information on Bonjour, including links to standards, specifications, and resources, see http://developer.apple.com/networking/bonjour and read *Bonjour Overview*.

Documentation on the Java-based DNS Service Discovery API is available at http://developer.apple.com/documentation/Java/Reference/DNSServiceDiscovery_JavaRef/index.html.

For information on dynamic update and shared secrets, see http://www.ietf.org/rfc/rfc2136.txt and http://www.ietf.org/rfc/rfc2845.txt, respectively.

For information Network Address Translation (NAT), see http://www.ietf.org/rfc/rfc3022.txt, and for information on automatic NAT port mapping, see http://files.dns-sd.org/draft-nat-port-mapping.txt.

2005-11-09 | © 2005 Apple Computer, Inc. All Rights Reserved.

Registering and Terminating a Service

When your service starts up, you need to register with the mDNSResponder daemon so that applications can discover your service. This section provides a general overview of the process, followed by a set of step-by-step instructions.

Registering a Service

To register your service, call `DNSServiceRegister`. The parameters for making this call consist of the following:

- An uninitialized service discovery reference.
- The index for the interface on which you want to register your service; pass `0` to register on all available interfaces, pass `-1` to register on the local machine only (your service will not be available to remote hosts), or pass the number that represents the interface on which you want to register (use the `if_nametoindex` family of calls to get the number).
- Flags that indicate how you want to handle name conflicts. By default, `(`*n*`)` is automatically appended to your service name, where *n* is a number, if a name conflict occurs. To override this behavior, set the `kDNSServiceFlagsNoAutoRename` flag, which will cause your registration callback function to be called so that you can handle name conflicts. The `kDNSServiceFlagsNoAutoRename` flag is only valid if you also explicitly specify a service name.
- The service's name; you can specify `NULL` to use the computer's name as the service's name.
- The service's registration type.
- The SRV target host name; usually, you'll pass `NULL` to use the computer's default host name. Passing `NULL` is the desired behavior in almost every case. However, proxy applications may pass an explicit SRV target other than the computer's host name.
- The port number in network byte order on which the service accepts connections. Passing `0` for the port registers a placeholder service. With a placeholder service, the service will not be discovered by browsing, but a name conflict will occur if another client tries to register the same name. Most applications do not need to use placeholder service.
- The callback function that is to be called to provide information on the success or failure of the registration, or `NULL`.
- A user-defined context value that will be passed to the callback function when it is called, or `NULL`.

Services that require TXT records also pass the raw data of the TXT record and the length of the raw data as parameters. Most services don't need TXT records and therefore pass `NULL` and `0`, respectively, for these parameters.

Instead of providing a callback function, you may pass `NULL`, in which case, you will not be notified of default values that may be chosen on your behalf and you will not be notified of any asynchronous errors that may prevent the registration of your service. If you pass `NULL` for this parameter, you cannot pass `kDNSServiceFlagsNoAutoRename` as the flag parameter. You can de-register a service that is registered without a callback function in the normal way, by calling `DNSServiceRefDeallocate`.

If the registration can be started, `DNSServiceRegister` initializes the service discovery reference and creates a socket that is used to communicate with the mDNSResponder daemon. Use the service discovery reference to call `DNSServiceRefSockFD` and get the socket descriptor for the service reference.

Set up a run or `select` loop using the socket descriptor. When the loop indicates that the mDNSResponder daemon's reply is available, call `DNSServiceProcessResult` and pass to it the service discovery reference initialized by `DNSServiceRegister`. `DNSServiceProcessResult` will call the callback function associated with the service discovery reference.

Instead of setting up a run loop or a `select` loop, you can call `DNSServiceRegister` and immediately call `DNSServiceProcessResult`. The `DNSServiceProcessResult` function will block until the mDNSResponder daemon has a response, at which time the callback specified when `DNSServiceRegister` was called (if any) will be invoked.

In addition to the service discovery reference and flags that are not currently used, your callback will be called with the following parameters:

- An error code that indicates whether the registration was successful; if the registration was successful, the remaining parameters contain valid data
- The service's name as passed to `DNSServiceRegister` or the name that was chosen if `NULL` was passed to `DNSServiceRegister` as the service's name
- The registration type as passed to `DNSServiceRegister`
- The domain in which the service was registered
- The user-defined context information that was passed to `DNSServiceRegister`

If the combination of service name, registration type, and domain name resulted in a full domain name that is already in local use and you specified `kDNSServiceFlagsNoAutoRename`, you'll need to deallocate the service discovery reference, choose another service name and try again, until a locally unique name can be registered.

Upon successful registration, your service is announced to the local network and its access information (IP address, port, and so on) can be found using multicast DNS, either by name or by browsing for services. Using the initialized service discovery reference, you can communicate with the mDNSResponder daemon to add a record to the registration information for your service, update an added record, or remove an added record while your service is running. However, you will probably never need to make changes to your registration information because Bonjour handles the common cases, such as waking, sleeping, shutting down, and changing IP addresses.

A rare exception would be the need to update the text record associated with a service. If a text field contains a queue name, for example, and the queue name changes, you would need to update the text record for the service.

You must keep the socket descriptor on the run loop or the `select` loop as long as you expect your callback function to be called.

Terminating a Service's Registration

To terminate your service's registration, remove the socket descriptor from the run loop or the `select` loop and call `DNSServiceRefDeallocate`, passing to it the service discovery reference that was initialized when your service was registered. In addition to invalidating the service discovery reference and deallocating the memory associated with it, any resource records that have been added are de-registered and their references are invalidated. The socket that underlies the connection with the mDNSResponder daemon is closed, thereby terminating your application’s connection with the daemon.

2005-11-09 | © 2005 Apple Computer, Inc. All Rights Reserved.

Browsing for Network Services

Browsing for services using this API is fairly simple. You can find out what services of a given type are available in a given domain with a single function call.

Using DNSServiceBrowse

To browse for available services, call `DNSServiceBrowse`. The parameters for making this call consist of the following:

- An uninitialized service discovery reference.
- The index for the interface you want to browse; pass `0` to browse all available interfaces, pass `-1` to browse for services on the local host only, or pass the number that represents the interface you want to browse (use the `if_nametoindex` family of calls to get the number).
- The registration type of the service you want to browse; the registration type is the service type followed by a dot, followed by the protocol (for example, `_printer._tcp`).
- The domain to browse; pass `NULL` to browse the domain(s) specified by the user as acceptable for browsing or pass a domain name to only browse that domain.
- The callback function that is to be called to provide information on the success or failure of the browse and to provide search results, such as a service that has been found or a service that is no longer available.
- A user-defined context value that will be passed to the callback function when it is called, or `NULL`.

If the browse can be started, `DNSServiceBrowse` initializes the service discovery reference and creates a socket that is used to communicate with the mDNSResponder daemon. Use the service discovery reference to call `DNSServiceRefSockFD` and get the socket descriptor for the socket.

Use the socket descriptor to set up a run loop or a `select` loop that will indicate when a response from the mDNSResponder daemon becomes available. The response may indicate that a service instance matching the specified type, domain, and interface has been found or that a service instance that was previously found is no longer available. When the loop indicates that the mDNSResponder daemon has responded, call `DNSServiceProcessResult` and pass to it the service discovery reference initialized by `DNSServiceBrowse`. `DNSServiceProcessResult` will call the callback function associated with the service discovery reference.

Your callback will be called with the following parameters:

- The service discovery reference that was initialized by `DNSServiceBrowse`.
- Flags that provide information about a service that has been found or that is no longer available and browsing status. For example, `kDNSServiceFlagsAdd` indicates that the service parameter contains the name of a service that has been found; you should add it to your list of available services. If `kDNSServiceFlagsAdd` is not set, the service specified by the service parameter is no longer available and should be removed from your list of available services. Browsing status is indicated by the `kDNSServiceFlagsMoreComing` flag. When it is set, your callback function will be called again

immediately, so you should not update your user interface. When `kDNSServiceMoreComing` is not set, your callback function will not be called again immediately, so you have time to update your user interface.

- The index of the interface on which the service was discovered.
- An error code that indicates whether browsing was successful; if browsing was successful, the remaining parameters contain valid data.
- The name of the service that was found, if browsing was successful.
- The registration type, if browsing was successful.
- The domain in which the service was discovered, if browsing was successful.
- The user-defined context information that was passed to `DNSServiceBrowse`.

Browsing Multiple Domains

To browse in multiple domains, or for multiple service types, call `DNSServiceBrowse` for each domain and service type of interest. Your application is responsible for keeping track of the responses.

Note: You can obtain a list of recommended domains to search by calling `DNSServiceEnumerateDomains`. For details, see "Enumerating Domains" (page 21).

If your application needs to leave the browser interface visible the entire time your application is running, as iTunes and iChat do, then you typically will call `DNSServiceBrowse` once per session. Whenever a new service becomes available or an existing service becomes unavailable, data is sent to your callback function, so you can simply leave the callback active, and your list of services will always be up to date. This information typically changes infrequently, so the callback shouldn't use much CPU time.

However, if you application does not need to constantly show the list of available services, in a situation such as the printer dialog, then you should call DNSServiceBrowse and terminate the browsing when you are finished.

When you call `DNSServiceBrowse`, it initializes a service discovery reference and opens a socket-based connection with the mDNSResponder daemon. For this reason, if you choose to deactivate your callback and repeat the search as needed, be sure to call `DNSServiceRefDeallocate` to deallocate the reference before calling `DNSServiceBrowse` again. Otherwise, you will leak memory and sockets for every search.

The actual IP address and port of a given service instance will change more frequently than the service name. Each time you use the service, you should resolve the current address of a service instance just prior to using the service. See the next section, "Resolving the Current Address of a Service" (page 17).

Terminating Browsing

To terminate browsing, remove the socket descriptor from the run loop or the `select` loop and call `DNSServiceRefDeallocate`, passing to it the service discovery reference that was initialized when `DNSServiceBrowse` was called. Browsing is halted, the service discovery reference is invalidated, and memory associated with the reference is deallocated. The socket that underlies the connection with the mDNSResponder daemon is closed, thereby terminating your application's connection with the daemon.

Resolving the Current Address of a Service

This article describes how to use `DNSServiceResolve` to get information about a service based on its name, type, and domain.

Using DNSServiceResolve

Once you have the name, registration type, and domain of a service, you can get information about the service, such as the interface(s) on which the service is registered, the full domain name of the service, name of the host that provides the service, and the content of the service's primary TXT record, by calling `DNSServiceResolve`.

Warning: `DNSServiceResolve` is appropriate for getting information about a service that has a single SRV record and a single TXT record (which may be empty). To resolve services that have multiple SRV or TXT records, you should use `DNSServiceQueryRecord` You should also use `DNSServiceQueryRecord` to monitor TXT record content instead of `DNSServiceResolve`.

To resolve a service name to its hostname and port, call `DNSServiceResolve`. The parameters for making this call consist of the following:

- An uninitialized service discovery reference
- The index of the interface on which you want to resolve the service; pass the value that was passed to your callback function for `DNSServiceBrowse`, or `0` to resolve on all available interfaces
- The service name to be resolved; pass a value that was passed to your callback function for `DNSServiceBrowse`
- The registration type of the service to be resolved; pass the value that was passed to your callback function for `DNSServiceBrowse`
- The domain in which the service is registered; pass the value that was passed to your callback function for `DNSServiceBrowse`
- The callback function that is to be called to provide information on the success or failure of the resolution
- A user-defined context value that will be passed to the callback function when it is called, or `NULL`

If the resolution can be started, `DNSServiceResolve` initializes the service discovery reference and creates a socket that is used to communicate with the mDNSResponder daemon. Use the service discovery reference to call `DNSServiceRefSockFD` and get the socket descriptor for the socket.

Setting Up a Callback Function

If `DNSServiceResolve` returns error-free, you need to have mDNSResponder resolve the service discovery reference and run a callback function when it has received a response. There are two techniques to set up the callback function: asynchronously and synchronously.

To get a response from mDNSResponder asynchronously, set up a run or a `select` loop using the socket descriptor. The loop will be notified whenever a response from the mDNSResponder daemon becomes available. When the loop indicates that a response is available, call `DNSServiceProcessResult` and pass to it the service discovery reference initialized by `DNSServiceResolve`. `DNSServiceProcessResult` will call the callback function associated with the service discovery reference. The mDNSResponder daemon will provide a response for each service that it resolves on a per-interface basis.

If you want to run the callback function synchronously instead of setting up a run loop or a `select` loop, you can call `DNSServiceResolve` and immediately call `DNSServiceProcessResult`. The `DNSServiceProcessResult` function will block until the mDNSResponder daemon has a response, at which time the callback specified when `DNSServiceResolve` was called will be invoked. This entire process should probably be run within a loop of its own for each service you wish to resolve.

In addition to the service discovery reference and flags that are not currently used, your callback will be called with the following parameters:

- The interface index on which the service was resolved; use the `if_nametoindex` family of calls to relate the index to an interface name
- An error code that indicates whether the resolution was successful; if the resolution was successful, the remaining parameters contain valid data
- The full domain name of the service, suitable for passing to special purpose functions that take a full domain name as a parameter
- The hostname of the machine that provides the service, suitable for passing to `gethostbyname` or `DNSServiceQueryRecord` to get the host's IP address
- The port number in network byte order on which the service accepts connections
- The length of the TXT record for the service
- The primary TXT record for the service in standard TXT record format (that is, a length byte followed by data, followed by a length byte, followed by data, and so on)
- The user-defined context information that was passed to `DNSServiceResolve`

Important: The service's IP addresses and port numbers can change dynamically, so you should get the current address each time you use a service, just prior to using it.

Your run loop or `select` loop will be notified for each interface on which the service is resolved and for each TXT record associated with the service.

When the desired results have been obtained, you must terminate the resolution. Remove the socket descriptor from the run loop or the `select` loop and call `DNSServiceRefDeallocate`, passing to it the service discovery reference that was initialized when `DNSServiceResolve` was called. The service discovery reference is invalidated, and memory associated with the reference is deallocated. The socket that underlies the connection with the mDNSResponder daemon is closed, thereby terminating your application's connection with the daemon.

2005-11-09 | © 2005 Apple Computer, Inc. All Rights Reserved.

Enumerating Domains

The `DNSServiceEnumerateDomains` function finds domains that are recommended for registration and browsing. Each time your callback is called, information about one domain is provided, along with flags indicating whether to add or remove the domain from your list of domains or indicating that the domain is a default domain, or is no longer the default domain.

The parameters for calling `DNSServiceEnumerateDomains`consist of the following:

- An uninitialized service discovery reference
- A flag that indicates whether you want to enumerate recommended browsing or registration domains
- An interface index that specifies the interface to enumerate; pass `0` to enumerate domains on all interfaces or a positive integer to specify the interface on which to enumerate domains (use the `if_nametoindex` family of calls to get the index of the interface you want to enumerate)
- The callback function that is to be called to provide information on the success or failure of the enumeration
- A user-defined context value that will be passed to the callback function when it is called, or `NULL`

If the enumeration can be started, `DNSServiceEnumerateDomains` initializes the service discovery reference and creates a socket that is used to communicate with the mDNSResponder daemon. Use the service discovery reference to call `DNSServiceRefSockFD` and get the socket descriptor for the socket.

Set up a run loop or a `select` loop using the socket descriptor. When the loop indicates that a response from the mDNSResponder daemon is available, call `DNSServiceProcessResult` and pass to it the service discovery reference initialized by `DNSServiceEnumerateDomains`. `DNSServiceProcessResult` will call the callback function associated with the service discovery reference.

Instead of setting up a run or `select` loop, you can call `DNSServiceEnumerate` and immediately call `DNSServiceProcessResult`. The `DNSServiceProcessResult` function will block until the mDNSResponder daemon has a response, at which time the callback specified when `DNSServiceEnumerate` was called will be invoked.

Your callback will be called with the following parameters:

- The service discovery reference that was passed to `DNSServiceEnumerateDomains`
- Flags that indicate whether your callback will be called again immediately to pass information about another domain that has been found, whether to add or remove this domain from the list that your application maintains, and whether to add or remove the domain as a default domain
- The index of the interface on which the domain was found
- An error code that indicates whether the enumeration was successful; if the enumeration was successful, the other parameters contain valid data
- The name of the domain that was found
- The user-defined context information that was passed to `DNSServiceEnumerateDomains`

The run loop or the `select` loop will be notified for each recommended domain enumerated on per-interface basis and whenever a domain is added or removed. You are responsible for assembling the daemon's responses into a list of current recommended domains.

Note: Even if the flag indicates that the list is complete, your callback will be called again if a domain is added or removed, made the default, or is no longer the default.

To terminate the enumeration, remove the socket descriptor from the run loop or the `select` loop and call `DNSServiceRefDeallocate`, passing to it the service discovery reference that was initialized when `DNSServiceEnumerateDomains` was called. The service discovery reference is invalidated, and memory associated with the reference is deallocated. The socket that underlies the connection with the mDNSResponder daemon is closed, thereby terminating your application's connection with the daemon.

Using DNS Service Discovery in Windows

DNS Service Discovery was written with cross-platform compatibility in mind. Therefore, all of the DNS Service Discovery API calls that are valid in Mac OS X are also valid in Windows. The difference between the two platforms lies in how each handles run loops. The next two sections will explain what changes need to be made to write programs that take advantage of DNS Service Discovery in Windows. Before reading these sections, you'll want to become familiar with the DNS Service Discovery API and Microsoft Foundation classes, if you are not already.

Windows Graphical User Interfaces

To properly incorporate DNS Service Discovery in a Windows graphical user interface, use the WinSock API WSAAsyncSelect. WSAAsyncSelect integrates socket-based network events into the Windows message loop. To use this in your Windows code, you should first create and initialize a DNSServiceRef. Then, call the function `WSAAsyncSelect` to associate your DNSServiceRef's socket with the Windows message loop. `WSAAsyncSelect` requires four arguments: a socket to your DNSServiceRef, a window to receive the message, a message to be sent when the event occurs, and a bitmask for the network events you are interested in. A simple example of this is provided below. In the example, you can see how to create a `NULL` DNSServiceRef, initialize that reference with `DNSServiceBrowse`, and then add it to the work loop with `WSAAsyncSelect`.

```
// create blank DNSServiceRef
e = new ServiceHandlerEntry;
...
// initialize the DNSServiceRef for browsing
err = DNSServiceBrowse( &e->ref, 0, 0, e->type, NULL, BrowseCallBack, e );

// add browsing to the work loop with WSAAsyncSelect
//  where m_hWnd is the window, WM_PRIVATE_SERVICE_EVENT is the message and
//  FD_READ and FD_CLOSE are bitmasks for reading and closing sockets
err = WSAAsyncSelect((SOCKET) DNSServiceRefSockFD(e->ref),
                    m_hWnd,
                    WM_PRIVATE_SERVICE_EVENT,
                    FD_READ|FD_CLOSE);
```

Windows Command-Line Interfaces

Creating a Windows command-line program using DNS Service Discovery is similar to creating one for Mac OS X. Windows, like Mac OS X, has support for the `select` system call. This function is used to determine when results are available from the DNS Service Discovery API functions. More information about using the `select` loop with DNS Service Discovery is available in "Registering and Terminating a Service" (page 9), "Browsing for Network Services" (page 13), and "Resolving the Current Address of a Service" (page 17).

2005-11-09 | © 2005 Apple Computer, Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *DNS Service Discovery Programming Guide*.

Date	Notes
2005-11-09	Changed title from "DNSServiceDiscovery Programming Guide."
2005-10-04	Updated domain information.
2005-06-06	Added Windows-specific information.
2005-04-29	Updated for Mac OS X v10.4. Changed "Rendezvous" to "Bonjour." Changed title from "DNSServiceDiscovery API."
2004-02-01	First version of this document, which describes the socket-based API that replaces the Mach-based DNS Service Discovery API.