Programmer's Guide and Reference

Agent API

The Team:
Engineering:
Marketing:
Publications:
Quality Assurance:
Technical Support:

Version 1.0

Printed in USA
99 98 97 10 9 8 7 6 5 4 3 2 1

Netscape Communications Corporation 501 East Middlefield Road, Mountain View, CA 94043

Contents

1

Using the Agents API

Introduction

The Enterprise Server supports a C-language API that you can use to write applications that create, modify, and delete agents. This API is a server plug-in API. (A server plug-in API is a set of functions and header files that help you create functions to use with the directives in the `obj.conf` server configuration file.)

Note that the Agents API cannot be used for CGI development or for developing out-of-process WAI applications. (The Agents API functions all run within the web server process.)

All functions in the Agents API are thread-safe.

This section covers the following topics:

- Who Should Read This Guide?
- What's In This Guide?

Who Should Read This Guide?

This manual is intended for C programmers who want to write server plug-in applications to create and modify agents for the Enterprise Server. Readers of this manual should already be familiar with server plug-in APIs.

Important If you are not familiar with server plug-in APIs, you should read the online manual *Netscape Enterprise Server Programmer's Guide 2.0* for Windows NT and UNIX, which are available from the Netscape DevEdge Online web site.

What's In This Guide?

This manual describes how to use the agents API to write server plug-in applications that create and modify agents in the Enterprise Server. This manual consists of two parts:

- **Using the Agents API.** This part contains a guide to using the agents API.
- **Reference.** This part is a reference section covering each function, data type, and error code in the API.

A more detailed description of the contents of this manual is listed below.

Part I: Using the Agents API

"Introduction"	Contains information about this manual and introductory material about the agents API.
“Understanding Agents and the Agents API”	Provides background information on agents and on how they work in the Enterprise Server.
“Writing Plug-ins with the Agents API”	Contains general instructions on writing and compiling server plug-ins that use the agents API.
“Creating A New Agent”	Explains how to create a new agent, including steps for: • Creating a new agent object • Registering the agent • Freeing the agent object from memory
“Working With Existing Agents”	Explains how to work with registered agents, including steps for: • Finding agents • Getting and changing data for agents • Changing the state of agents • Working with agent IDs

“Working With Events”

Explains how to create and modify events, including steps for:

- Creating events
- Specifying the class of an event
- Specifying the message ID of an event
- Setting the context of a document-based event
- Setting the context of a timer-based event
- Freeing events from memory

“Working With Actions”

Explains how to create actions that send mail messages and post messages to newsgroups, including steps for:

- Creating actions for mailing and posting to newsgroups
- Creating a list of actions
- Creating an action
- Specifying the class of an action
- Specifying information about the action
- Adding actions to an action list
- Getting an action from an action list
- Counting actions in an action list
- Freeing actions and action lists from memory

Part II: Reference

“Data Types”	Describes the data types in this API.
“API Function Reference”	Describes each function in this API.
“Error Code Reference”	Describes each error code that can be returned by an API function.

Chapter 1

Understanding Agents and the Agents API

In some situations, you may want the Enterprise Server to perform an action automatically without requiring you to directly interact with the server. For example, you may want to get a list of the most recently updated web pages without having to start your browser, access the server, and perform a search.

In the Enterprise Server, user and administrators can set up **agents** that can perform actions automatically without requiring direct user interaction. (Sometimes, these are referred to as **server-side agents**.)

The Enterprise Server provides a standard forms-based interface for creating and managing agents (through the Agent Services page) as well as a programmatic interface (through the Agents API).

Use of the forms-based interface is explained in the online manual *Web Publisher User's Guide*, which is provided with the Enterprise Server. Use of the Agents API is described in this manual. The rest of this section explains the concepts of agents in terms of the Agents API.

Understanding How Agents Work

An **agent** is an object within a server that performs various requests (such as HTTP, NNTP, and SMTP requests) on behalf of the user. In a sense, the agent acts as a client to the server, making requests that the server fulfills.

Agents work in the following way:

1. A user defines an agent that interacts with the server to perform some **actions** when a specific **event** occurs.

   For example, the user might define an agent that requests a particular URL when the document identified by that URL is modified. The event occurs when the document is modified. The action that is invoked is that the agent sends an HTTP GET request to get the document from the server.

2. Once the agent is created, the agent keeps track of the user's request and performs the specified actions whenever the event occurs.

Two important components that make up an agent are:

- an event
- a set of actions

An **event** is something that triggers the actions. In general, there are two types of events:

- **Document events.** These events are based on the state or properties of a document or directory served by the Enterprise Server. For example, an event might be the removal of a directory or the modification of a document.

- **Timer events.** These events are based on a specific point in time (for example, Monday, June 30, 1997 at 12PM) or a recurring period of time (every day, every week, every month).

An **action** (or a set of actions) is a task performed by the agent when the event occurs. The kinds of actions that an agent can perform include:

- Sending an email message
- Posting a message to a newsgroup
- Sending an HTTP GET, POST, or PUT request
- Logging a message to a file
- Running a program

You can get a better understanding for how these work by using the Agent Services page in the Enterprise Server to create and modify agents. (In that forms-based interface, Document Agents and Directory Agents correspond to agents based on document events, and Timer Agents and Search Agents correspond to agents based on timer events.)

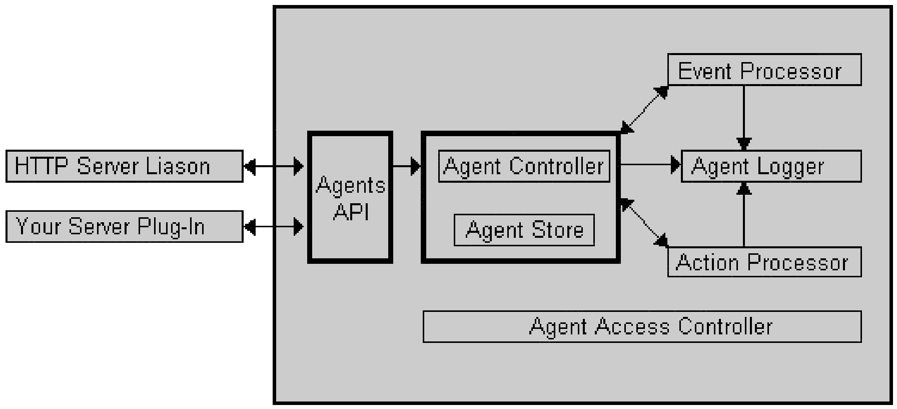
Understanding How the Agents API Works

In the Agents API, agents, events, and actions are represented by opaque structures that you can create, manipulate, and destroy. You call API functions to create new agent objects, manipulate existing agent objects, and specify the events and actions associated with agents.

Although you can specify the events and actions associated with an agent through the Agents API, you cannot directly affect the execution and processing of the events and the actions themselves.

The following figure illustrates how the Agents API relates to the different components of the agent framework in the server.

Figure 1.1



Note that most of these components are internal and are not directly accessible through the API (for example, the Action Processor is not directly accessible through the API, so you cannot control the way in which actions are processed).

The components of the agent framework are described below. The internal components (which are not accessible through the Agents API) are described in order to provide a complete perspective of how the system operates and to delineate the scope of the API more clearly.

- **Agent Controller (AC) component**

The Agent Controller (AC) component is the heart of the system. It manages all aspects of agent creation, destruction, statistics update, persistent storage, and parts of agent administration.

The agent framework provides two interfaces to this component:

- Programmers can develop server plug-ins using the Agents API. This API provides access to creating, manipulating, and destroying agent objects.
- Users of the Enterprise Server can create and manage agents through a web-based interface through the HTTP Server Liason (see next item).

The AC component distinguishes each agent by assigning it a unique identification number (an Agent ID). This ID is also used to save and retrieve the agent from the Agent Store (see below). Agent IDs are unique within a server but not across servers.

- **HTTP Server Liaison (HSL) component**

The HTTP Server Liaison (HSL) allows the AC to communicate with an HTTP server. This front end the AC to present users with a web-based interface (such as HTML forms) for creating and managing agents. For example, users of the Enterprise Server can use the forms-based interface provided with the server to create and manage their own agents.

The HSL component is responsible for accepting the HTTP input and parsing it into appropriate arguments, which are then passed to the AC.

HSL also enforces some preliminary access control issues. Based on the request coming in, it can check if the user is authorized to access agent services.

Note The HSL component is not used in plug-in development. It is listed here as a component of the agents system to illustrate how the existing forms for creating and managing agents fit into the system.

- **Agent Store component**

The Agent Store component manages the persistent storage of agent information. The AC uses the agent store to save and retrieve agents.

When you create an agent through the Agents API, you need to register the agent with the Agent Store. Part of this process records the agent information in the Agent Store.

When you work with existing agents, the AC finds and reads the agents from the Agent Store, using the Agent ID to identify specific agents. Any changes that you decide to make to the agent object in memory need to be saved to the Agent Store.

The Agents API includes functions for saving modified agents to the Agent Store and for deleting agents from the Agent Store.

- **Event Processor (EP) component**

The Event Processor component manages event detection and triggering of agents. The EP is responsible for understanding different event types, registering for them, detecting them and finally triggering agents.

The Event Processor is an internal component that is not directly accessible through the API.

- **Action Processor (AP)**

The Action Processor component is responsible for performing the actions (such as sending mail, POSTing to a page, running a program) when an agent is triggered in response to some event. Examples of actions include:

  - Use an HTTP GET request to retrieve a document and mail the document to a user
  - Use an HTTP POST request to post data to a URL
  - Search a collection for recently modified documents
  - launch a script or a program

In principle, the Action Processor behaves as a client to the web server, sending requests that use either the HTTP, SMTP or NNTP protocol.

The Action processor receives a stream of requests from the Agent Controller to execute actions for various agents. These actions are executed in a concurrent fashion (multithreading). There is no mechanism to enforce a specific order of execution among agents, and there is no inter-agent communication available. (Agents cannot exchange data or interact with each other.)

- **Agent Logger**

The Agent Logger component is responsible for logging all necessary information in an external log file. Other components call on the Agent Logger with information to be logged.

This is another internal component that is not directly accessible through the Agents API.

- **Agent Access Controller**

The Agent Access Controller module takes care of validating ACL for any operation performed by clients. It is responsible for maintaining ACL and such to validate access.

Chapter

2

Writing Plug-ins with the Agents API

The agent API functions allow you to create, modify, and delete agents, events, and actions. For example, you can call API functions to change the agent's expiration date or the list of actions associated with the agent.

The Agents API is a server plug-in API. You need to be familiar with server plug-ins before using this API.

Important If you are not familiar with server plug-in APIs, you should read the online manual *Netscape Enterprise Server Programmer's Guide 2.0* for Windows NT and UNIX, which are available from the Netscape DevEdge Online web site.

Keep the following tips in mind if you are using the Agents API functions in your server plug-in.

- Include the `agent_c_api.h` header file:

```
#include <agent_c_api.h>
```

This file declares the agents API functions. You can find this header file in the server root directory under `include/agents` for UNIX and `include\agents` for Windows NT.

- To avoid problems with name mangling, use a native compiler or Microsoft Visual C++.

The Agents API shared library was compiled on each platform using the native compiler for that platform (for example, Sun's CC compiler was used to compile the `agents.so` shared object for Solaris) and with Visual C++ on Windows NT.

- Link to the agents API library.
    - On UNIX, you can find the library under the server root at `plugins/agents/bin/agents.so`. (Note that the filename extension will vary, depending on the UNIX platform.)
    - On Windows NT, you can find the library under the server root at `plugins\agents\lib\agents.lib` and the corresponding DLL under `plugins\agents\bin\agents.dll`.
- After compiling your plug-in, you need to configure the Enterprise Server to load and initialize your plug-in. In the `obj.conf` file, add the following directive:

```
Init fn="load-modules" funcs="myInitFunction" shlib="abs_path_to_plug_in"
Init fn="myInitFunction"
```

where *`myInitFunction`* is the name of your initialization function and *`abs_path_to_plug_in`* is the absolute path to your plug-in.

For example, if you have defined the initialization function `agenttest_init` in the shared library `myplugin.so`, your initialization directives in the `obj.conf` would look like this:

```
Init fn="load-modules" funcs="agenttest_init" shlib="/usr/netscape/suitespot/plugins/
myplugin.so"
Init fn="agenttest_init"
```

Chapter

3

Creating A New Agent

In the API, an agent is represented by an agent object (an object with the data type `ns_agent_t`). Information specified by an agent object includes:

- The event that should trigger the specified actions
- A list of the actions that should be triggered when the event occurs
- Name and email address of the user for whom the agent is working
- Maximum number of triggers and expiration date for the agent

For more information about creating events and lists of actions, see "Working With Events" and "Working With Actions".

To create a new agent, you need to:

1. Create a new agent object.
2. Register the new agent object.
3. Free the agent object from memory when done.

See the example of creating a new agent at the end of this section.

Creating a New Agent Object

To create a new agent object, call the `ns_AgentCreateAgentObject()` function. This function returns a value of the `ns_agent_t` type, which represents the newly created agent object. Pass this value as a parameter to other functions to manipulate the agent object.

Use the parameters of the `ns_AgentCreateAgentObject()` function to specify information about the agent.

Note Two of the arguments that you can specify are the event object and the action list object, which represent the event and the list of actions that you want associated with this agent. Once you pass these objects as arguments to the `ns_AgentCreateAgentObject()` function, you should not free the event object or the action list object from memory. These objects become adopted as part of the agent object.

When you call the `ns_AgentDeleteAgentObject()` function to free the agent object from memory, the event object and the action list object are also freed from memory.

If you want to specify any of the agent information later, you can pass `NULL` values for the arguments in `ns_AgentCreateAgentObject()` and use the other API functions to set the data in the agent object. For details, see the section "Getting and Setting Data in Agent Objects".

When you are done working with the agent object, you need to free the object from memory. For details, see "Freeing Unused Agent Objects from Memory".

Registering New Agent Objects

After you fill the values of the agent object's fields, you can register the new agent object. To register a agent, call the `ns_AgentSubmit()` function. If successful, the function returns `NS_AGENT_SUCCESS`.

When you register the agent object, the Agent Controller does the following:

1. Sets the state of the new agent to "enabled".
2. Records the creation date and time of the agent. (The Agent Controller also sets the "last modified" time of the agent to the current time.)

3. Saves the agent to the Agent Store.

When an agent is added to the Agent Store, the agent is assigned an agent ID. To find and retrieve the agent data from the Agent Store in the future, you need to specify this ID. For information on getting the agent ID for the agent object, see Getting the ID for an agent object.

Freeing Unused Agent Objects from Memory

When you are done working with an agent object, you should free the object from memory by calling the `ns_AgentDeleteAgentObject()` function.

Note This function does not remove the agent from the Agent Store; the function simply frees the existing agent object from memory.

Example: Creating a New Agent

The following example illustrates how to create a new agent.

```
...
#include <agent_c_api.h>
...
ns_status_t st;
ns_agent_t agent;
ns_agentEvent_t event;
time_t expiryDate;
size_t triggerLimit;
char *user = "mozilla";
char *addr = "mozilla@netscape.com";
ns_agentActionList_t actionList;
char *desc = "My New Agent";
...

/* Set up the agent to be deleted after being triggered 100 times
   or after one day */
triggerLimit = 100;
expiryDate = time(NULL) + 86400;

...
```

```
/* Code for creating the events and actions goes here. */
...

/* Create a new agent object and specify information about the agent */
agent = ns_AgentCreateAgentObject(user, addr, expiryDate, triggerLimit,
                                  0, event, actionList, NULL, 0, desc);

/* Register the new agent object */
st = ns_AgentSubmit(agent);

/* Free the agent object from memory when done.
   NOTE: This will also free the event object and the
   action list object that you used to create this agent. */
ns_AgentDeleteAgentObject(agent);
...
```

Chapter

4

Working With Existing Agents

This section explains how to work with agents that are already registered in the Agent Store. Topics covered in this section include:

- Finding Agents By Event or User
- Getting the Agent Object from the Agent Store
- Getting and Setting Data in Agent Objects
- Changing the State of the Agent
- Working with Agent IDs

The end of the section contains an example of modifying an agent and changing the state of an agent.

Finding Agents By Event or User

To access and manipulate data in existing agents, you need to find and retrieve the agents from the Agent Store. To find an agent in the Agent Store, you need to specify the agent ID for that agent.

To get a list of the agent IDs for agents matching a specified criteria, call one of the following API functions:

- To list the IDs for the agents that monitor a specific event, call the `ns_AgentGetAgentListByEvent()` function.

- To list the IDs for the agents that work on behalf of a particular user, call the `ns_AgentGetAgentListByUser()` function.

After you are done with the agent IDs, you must free the list and the agent IDs from memory by calling the `ns_AgentDeleteAgentIdList()` function.

Getting the Agent Object from the Agent Store

Once you have the agent IDs for the agents that you plan to work with, you need to get the agent objects for those agents. (Note: If you only plan to change the state of the agent or to remove the agent, you do not need to retrieve the agent object. You just need the agent ID to perform these tasks.)

To get the agent object from the Agent Store, call the `ns_AgentLookup()` function. If successful, the function returns `NS_AGENT_SUCCESS`.

When you are done working with the agent object, you need to free the object from memory. For details, see "Freeing Unused Agent Objects from Memory".

Getting and Setting Data in Agent Objects

After you've created a new agent object or retrieved an agent object, you can call API functions to set and retrieve data from the agent object. The following table lists the functions that you can call to manipulate data in the agent object:

To get the creation date of the agent	`ns_AgentGetCreationDate()`
To get the date when the agent was last modified	`ns_AgentGetLastModifiedDate()`
To get the date when the agent was last triggered	`ns_AgentGetLastTriggerDate()`
To get the name of the user who the agent works for	`ns_AgentGetUserName()`

To get or set the name of the agent (**Note:** You can set the name of agents that have not been registered. Once the agent has been registered, you cannot change the agent's name by calling this function.)	`ns_AgentGetDescription()` `ns_AgentSetDescription()`
To get or set the expiration date of the agent	`ns_AgentGetExpiryDate()` `ns_AgentSetExpiryDate()`
To get or set the maximum number of times the agent can be triggered	`ns_AgentGetTriggerLimit()` `ns_AgentSetTriggerLimit()`
To determine the number of times the agent was triggered	`ns_AgentGetTriggerCount()`
To get or set the email address used to notify the agent's creator that the agent has been created	`ns_AgentGetNotificationAddr()` `ns_AgentSetNotificationAddr()`
To get or set the event monitored	`ns_AgentGetEventInfo()` `ns_AgentSetEventInfo()`
To get or set the list of actions invoked	`ns_AgentGetActionList()` `ns_AgentSetActionList()`
To get the number of actions in the list of actions	`ns_AgentGetActionCount()`

To save any changes you make to the Agent Store, call the `ns_AgentModify()` function. If successful, the function returns `NS_AGENT_SUCCESS`.

Changing the State of the Agent

An agent can be in one of a different number of states. To determine the current state of an agent, call the `ns_AgentGetState()` function. The function returns one of the following enumerated values to identify the state of the agent:

Agent State	Description
`eEnabled`	The agent is currently active and is capable of responding to events.
`eDisabled`	The agent is currently suspended and is not responding to any events.
`eDeleted`	The agent has been marked for removal from the server.

To change the state of the agent, call one of the following functions:

- To suspend (or disable) an agent, call the `ns_AgentDisable()` function.
- To reactivate a suspended agent, call the `ns_AgentEnable()` function.
- To remove an agent, call the `ns_AgentDelete()` function.
  (**NOTE:** You cannot undo the removal of an agent. If you want to restore an agent after removing it, you need to create the agent again.

Working with Agent IDs

When an agent is added to the Agent Store, the agent is assigned an agent ID, which is represented by the `ns_agentId_t` data type. You use the agent ID to identify an agent in the Agent Store.

To get the agent ID of an agent, call the `ns_AgentGetId()` function. The function returns the ID of the agent object.

If you want to convert this agent ID in a readable format, you can call the `ns_AgentIdToString()` function to get a string representation of this agent ID. The string representation has the following format:

```
className_instanceId
```

To convert the string representation back to an agent ID, call the `ns_AgentIdFromString()` function.

When you are done working with agent IDs, you should free the agent IDs from memory. To do this, call the `ns_AgentIdDelete()` function.

Example: Working with Existing Agents

The following example changes the trigger limit of an agent and saves the change back to the Agent Store. The example also enables any disabled agents.

```
...
#include <agent_c_api.h>
...
ns_status_t st;
ns_agentId_t *agentListP;
ns_agent_t agent;
int count;
...
/* Get the list of agents belonging to the user "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

/* Get the agent object for the first agent in the list */
if ((st == NS_AGENT_SUCCESS) && count  0) {
    st = ns_AgentLookup(agentListP[0], &agent);

    if (st == NS_AGENT_SUCCESS) {

        /* Change the trigger limit to 100 */
        ns_AgentSetTriggerLimit(agent, 100);

        /* Save the change to the Agent Store */
        ns_AgentModify(agent);

        /* Enable the agent if it is disabled */
        if (ns_AgentGetState(agentP) == eDisabled) {
            st = ns_AgentEnable(agentListP[i]);
        }

        /* Free the agent object from memory when done */
        ns_AgentDeleteAgentObject(agent);
    }

    /* Free the list of agent IDs */
    ns_AgentDeleteAgentIdList(*agentListPP, count);
}
...
```

Chapter
5
Working With Events

In the agents API, events are represented by the opaque data type `ns_agentEvent_t`. Events can be either based on the state of a resource (document-based events) or based on a specific time or period of time (timer-based events.)

For example, an event might occur when a document is modified (document-based event) or on June 30, 1997 at 12:00 PM (timer-based event).

This section covers the following topics:

- Creating a Document-Based Event
- Creating a Timer-Based Event
- Getting and Setting Information About an Event
- Freeing events from memory

Creating a Document-Based Event

To create a document-based event, you need to do the following:

1. Create an event object.
   Call the `ns_AgentEventCreate()` function to create the event object.
2. Specify the context of the document-based event. This includes specifying:

- The URL of the document that is being tracked
- The physical path of the document being tracked
- The specific event that is being tracked (for example, the event might trigger when the document is opened or modified)

To specify this information, call the `ns_AgentEventSetDocumentContext()` function. Use the `MessageId` data type to specify the event being tracked.

The following example creates a document-based event. The event is triggered if the document specified by the URL `/doc/my_file.html` is moved.

```
 ns_agentEvent_t event;
 ...
 event = ns_AgentEventCreate();
if (event) {
    ns_AgentEventSetDocumentContext(event,  NS_MOVE_MSG, "/doc/my_file.html",
                                         "/usr/ns-home/https-myhost/doc/myfile.html");
}
 ...
```

Creating a Timer-Based Event

To create a timer-based event, you need to do the following:

1. Create an event object.
   Call the `ns_AgentEventCreate()` function to create the event object.
2. Specify the context of the timer-based event. This includes specifying:
   - The date and time when the event occurs
   - The period of time between events, if the event is recurring

To specify this information, call the `ns_AgentEventSetTimerContext()` function. Use the `ns_agent_timer_type` data type to specify the timer type (for example, specific time or periodic).

The following example creates a timer-based event. The event is triggered once a week.

```
 ns_agentEvent_t event;
 time_t start_time;
 short daysVec[] = {0, 0, 0, 0, 0, 0, 0};
 unsigned long period = 1;
 ns_agent_timer_type type = eTimerPeriodicWeeks;
 ...
 /* Insert code here to set start_time to the starting time. */
 ...
 event = ns_AgentEventCreate();
 if (event) {
ns_AgentEventSetTimerContext(event, type,
   start_time, daysVec, period);
 }
 ...
```

Getting and Setting Information About an Event

After you've created a new event or retrieved an event for an existing agent, you can call API functions to set and retrieve data from the event object. The following table lists the functions that you can call to manipulate data in the event object:

To get the event class	`ns_AgentEventGetClass()`
To get and set the message ID (which identifies the event tracked)	`ns_AgentEventGetMessageId()` `ns_AgentEventSetMessageId()` `ns_AgentEventGetMessageIdAsText()`
To get and set the context of a document-based event	`ns_AgentEventGetDocumentContext()` `ns_AgentEventSetDocumentContext()`
To get and set the context of a timer-based event	`ns_AgentEventGetTimerContext()` `ns_AgentEventSetTimerContext()`

Freeing events from memory

When you are done working with an event, you should free the event from memory. To free an event from memory, call the `ns_AgentEventDelete()` function.

Chapter 6

Working With Actions

To create and modify the actions associated with an agent, call the API functions for manipulating the opaque data types `ns_agentActionList_t` (which represents a list of actions) and `ns_agentAction_t` (which represents an individual action).

After you create these actions and lists of actions, you can associate the actions lists with an agent by calling the `ns_AgentSetActionList()` function.

This section describes how to:

- Create actions
- Working with Lists of Actions
- Freeing Actions and Lists of Actions from Memory

Creating actions

The agents API includes functions that allow you to create actions that send mail messages and post articles to newsgroups.

The following table lists the functions that you can call to create actions.

To Do This:	Call This Function:
Create an action that sends mail to a user	`ns_AgentActionMailerCreate()`
Create an action that posts a message to a newsgroup	`ns_AgentActionNNTPPostCreate()`
Create an action that gets a URL and mails it to a user	`ns_AgentActionGetAndMailCreate()`
Create an action that posts data to a URL and mails the results to a user	`ns_AgentActionPostAndMailCreate()`
Create an action that searches a specified collection and mails the results to a user or posts them to a newsgroup	`ns_AgentActionSearchCreate()`
Create an action that does a combination of actions listed above	`ns_AgentCreateCompositeAction()`

To define a list of actions for an agent, follow these steps:

1. Create a list of actions.
   Call the `ns_AgentActionListCreate()` function, which returns `ns_agentActionList_t`.

2. Create an action.
   Call the appropriate function listed above, which returns `ns_agentAction_t`.

   For example, to create an action that mails a message to a user, call the `ns_AgentActionMailerCreate()` function

3. Add the action to the list of actions.
   Call the `ns_AgentActionListAdd()` function to add the action to the list.

   Define any other actions that you want associated with the event, and add them to the list as well.

4. Associate the list of actions with the agent.
Call the `ns_AgentSetActionList()` function to associate the actions with the agent.

Working with Lists of Actions

When creating or modifying lists of actions, you can call the following API functions to manipulate the action list.

To get the list of actions associated with the agent	`ns_AgentGetActionList()`
To get an action from the list of actions	`ns_AgentActionListGet()`
To count the number of actions in a list	`ns_AgentActionListCount()`
To get the action's class	`ns_AgentActionGetClass()`
To get or set the user associated with the action	`ns_AgentActionGetUser()`, `ns_AgentActionSetUser()`
To get the mail message sent by the action (applies only to actions that send email)	`ns_AgentActionGetEMailMessage()`
To get the email address where the action sends a message to (applies only to actions that send email)	`ns_AgentActionGetEMailAddress()`

Freeing Actions and Lists of Actions from Memory

When you are done working with an action or list of actions, you should free it from memory.

- To free an action from memory, call the `ns_AgentActionDelete()` function.

- To free the list of actions from memory, call the `ns_AgentActionListDelete()` function.

2 *Reference*

Chapter 7

Data Types

dataBlock_t

The `dataBlock_t` data type specifies a block of data passed to or returned by some of the API functions.

```
#include <agent_c_api.h>
...
typedef struct {
    void*  data;
    size_t size;
} dataBlock_t ;
```

This structure has the following fields:

`data`	The data.
`size`	The size of the data.

EAgentActionClass

The `EAgentActionClass` enumerated type specifies the class of an action. The class of an action determines what the action does.

```
#include <agent_c_api.h>
...
```

```
enum EAgentActionClass {
    eDoNothing,
    eCompositeAction,
};
```

Data of this type can have one of the following values:

`eDoNothing`	Performs no action.
`eCompositeAction`	Creates a multipart MIME message out of the results from an HTTP GET, an HTTP POST, and a search. The MIME types of the resulting message parts must be text/html or text/plain. The completed MIME message is either mailed to a specified user or posted to a newsgroup.

See Also

`ns_AgentActionGetClass()`.

EAgentState

The `EAgentState` enumerated type specifies the state of an agent.

```
#include <agent_c_api.h>
...
enum EAgentState {
    eEnabled,
    eDisabled,
    eDeleted
};
```

Data of this type can have one of the following values:

`eEnabled`	The agent is currently enabled.
`eDisabled`	The agent is currently disabled. (To disable an agent, call the `ns_AgentDisable()` function. To reenable a disabled agent, call the `ns_AgentEnable()` function.)
`eDeleted`	The agent has been marked for deletion. (To delete an agent, call the `ns_AgentDelete()` function.)

See Also

`ns_AgentGetState()`.

eAgentSubType

The `eAgentSubType` enumerated type specifies the subtype of an agent. Subtypes are used to distinguish different agents of the same type apart.

For example, in the user interface, the Timer Agent and the Search Agent are listed as different kinds of agents, even though they share the same same type. Agent subtypes are used to distinguish these different agents apart.

```
#include <agent_c_api.h>
...
enum eAgentSubType {
   eDocSubType,
   eDirSubType,
   eTimerSubType,
   eSearchSubType
};
```

Data of this type can have one of the following values:

`eDocSubType`	The agent is a Document Agent.
`eDirSubType`	The agent is a Directory Agent.
`eTimerSubType`	The agent is a Timer Agent.
`eSearchSubType`	The agent is a Search Agent.

See Also

`ns_AgentGetSubType()`, `ns_AgentSetSubType()`.

ESearchQueryType

The `ESearchQueryType` enumerated type specifies the type of search that an action should perform. You specify one of the values of this type when calling `ns_AgentActionSearchCreate()` or `ns_AgentActionSearchParse()` to create or parse an action that searches for documents in a collection.

```
#include <agent_c_api.h>
...
typedef enum { eSearch_by_Mdate,
               eSearch_All
} ESearchQueryType;
```

Data of this type can have one of the following values:

`eSearch_by_Mdate`	The search only includes documents in the collection that have been modified since the agent was last activated.
`eSearch_All`	The search includes all documents in the collection.

See Also

`ns_AgentActionSearchCreate()`, `ns_AgentActionSearchParse()`.

MessageId

The `MessageId` data type specifies the message ID of an event. The message ID determines the specific event that is tracked by the agent.

```
#include <agent_c_api.h>
...
typedef short MessageId;
```

The following values are valid message IDs:

NS_DELETE_MSG	Document was deleted.
NS_ACCESS_MSG	Document was accessed.
NS_PUT_MSG	Document was modified through the HTTP PUT protocol.
NS_EDIT_MSG	Document was edited.
NS_SAVE_MSG	Document was saved.
NS_HEAD_MSG	Document was accessed through the HTTP HEAD protocol.
NS_POST_MSG	Document was accessed through the HTTP POST protocol.
NS_OPEN_MSG	Document was opened.
NS_CLOSE_MSG	Document was closed.
NS_READ_MSG	Document was read.
NS_WRITE_MSG	Document was written to.
NS_MOVE_MSG	Document was moved.
NS_QUERY_ATTR_MSG	Document attributes were queried.
NS_SET_ATTR_MSG	Document attributes were set.
NS_GET_PROPERTIES_MSG	Document properties were retrieved.
NS_SET_PROPERTIES_MSG	Document properties were set.
NS_COPY_MSG	Document was copied.
NS_UNLOCK_MSG	Document revision was unlocked.
NS_LOCK_MSG	Document revision was locked.
NS_RC_CHECKIN_MSG	Document revision was checked in.
NS_RC_CHECKOUT_MSG	Document revision was checked out.
NS_RC_GET_LOG_MSG	Document revision logs were accessed.
NS_RC_LABEL_MSG	Document revision was assigned a label.
NS_RC_GET_REV_MSG	List of document revisions was accessed.
NS_RC_DIFF_MSG	Document revision was compared ("diff") against another revision.
NS_DELETE_DIR_MSG	Directory was deleted.
NS_MAKE_DIR_MSG	Directory was created.

NS_INDEX_MSG	The contents of the directory were listed.
NS_MOVE_DIR_MSG	Directory was moved.
NS_RM_ALL_DIR_MSG	All directories were removed.
NS_QUERY_DIR_ATTR_MSG	Directory attributes were queried.
NS_SET_DIR_ATTR_MSG	Directory attributes were set.
NS_COPY_DIRMSG	Directory was copied.
NS_TIMER_MSG	Timer event occurred.
NS_VERIFY_LINK	Link report for the document was generated.

See Also

`ns_AgentEventGetMessageId()`, `ns_AgentEventSetMessageId()`, `ns_AgentEventGetDocumentContext()`, `ns_AgentEventSetDocumentContext()`.

ns_agent_t

The `ns_agent_t` data type represents an agent. Use this data type to create and modify agents.

```
#include <agent_c_api.h>
...
typedef void* ns_agent_t;
```

See Also

`ns_AgentCreateAgentObject()`, `ns_AgentDeleteAgentObject()`, `ns_AgentLookup()`, `ns_AgentSubmit()`, `ns_AgentModify()`,`ns_AgentGetState()`,`ns_AgentGetId()`.

ns_agentAction_t

The `ns_agentAction_t` data type represents an action. Use this data type to specify and modify actions associated with agents.

```
#include <agent_c_api.h>
...
typedef void* ns_agentAction_t;
```

See Also

`ns_AgentActionCreate()`, `ns_AgentActionDelete()`, `ns_AgentActionListGet()`, `ns_AgentActionMailerCreate()`, `ns_AgentActionPostAndMailCreate()`, `ns_AgentActionGetAndMailCreate()`, `ns_AgentActionNNTPPostCreate()`, `ns_AgentActionSearchCreate()`.

ns_agentActionList_t

The `ns_agentActionList_t` data type represents a list of actions. Use this data type to specify and modify lists of actions associated with agents.

```
#include <agent_c_api.h>
...
typedef void* ns_agentActionList_t;
```

See Also

`ns_AgentActionListCreate()`, `ns_AgentActionListDelete()`, `ns_AgentGetActionList()`, `ns_AgentSetActionList()`, `ns_AgentActionListAdd()`, `ns_AgentActionListGet()`, `ns_AgentActionListCount()`.

ns_agentEvent_t

The `ns_agentAction_t` data type represents an event. Use this data type to specify and modify events associated with agents.

```
#include <agent_c_api.h>
...
typedef void* ns_agentEvent_t;
```

See Also

`ns_AgentEventCreate()`, `ns_AgentEventDelete()`, `ns_AgentGetEventInfo()`, `ns_AgentSetEventInfo()`, `ns_AgentGetAgentListByEvent()`.

ns_agentEventClass_t

The `ns_agentEventClass_t` data type specifies the type of event.

```
#include <agent_c_api.h>
...
typedef int ns_agentEventClass_t;
```

Data of this type can have one of the following values:

`NS_AGENT_DOCUMENT_EVENT`	The event is based on the state of a document.
`NS_AGENT_TIMER_EVENT`	The event is based on a specific time or time period.

See Also

`ns_AgentEventGetClass()`.

ns_agentId_t

The `ns_agentId_t` data type specifies the ID of the agent. You use the agent ID to identify an agent in the agent store.

```
#include <agent_c_api.h>
...
typedef void* ns_agentId_t;
```

See Also

ns_AgentGetId(), ns_AgentIdDelete(), ns_AgentIdToString(), ns_AgentIdFromString(), ns_AgentLookup().

ns_status_t

The ns_status_t data type specifies an error code returned by one of the API functions.

```
#include <agent_c_api.h>
...
typedef enum {
    NS_AGENT_SUCCESS = 0,
    NS_AGENT_STORE_INIT_FAILED = NS_AGENT_ERROR_BASE + 1,
    NS_AGENT_STORE_CORRUPTED,
    NS_AGENT_STORE_INCONSISTENT,
    NS_AGENT_ENTRY_NOT_FOUND,
    NS_AGENT_ENTRY_ALREADY_EXISTS,
    NS_AGENT_CALLER_NOT_AUTHORIZED,
    NS_AGENT_USER_LIMIT_REACHED,
    NS_AGENT_INVALID_EVENT,
    NS_AGENT_INTERNAL_ERROR,
    NS_AGENT_ACTION_PARSE_ERROR
} ns_status_t;
```

For a description of these error codes, see “Error Code Reference”.

See Also

ns_AgentStatusToString().

ns_agent_timer_type

The ns_agent_timer_type data type specifies the type of timer used in a timer event.

```
#include <agent_c_api.h>
```

```
...
enum ns_agent_timer_type{
    eTimerSpecific,
    eTimerPeriodicMinutes,
    eTimerPeriodicHours,
    eTimerPeriodicDays,
    eTimerPeriodicWeeks,
    eTimerPeriodicMonths,
    eTimerPeriodicYears,
    eTimerSelective,
    eUnknown=-1
};
```

Data of this type can have one of the following values:

`eTimerSpecific`	The event occurs at a specific time (for example, June 30, 1997 at 12:00 PM).
`eTimerPeriodicMinutes`	The event occurs periodically, every *X* minutes.
`eTimerPeriodicHours`	The event occurs periodically, every *X* hours.
`eTimerPeriodicDays`	The event occurs periodically, every *X* days.
`eTimerPeriodicWeeks`	The event occurs periodically, every *X* weeks.
`eTimerPeriodicMonths`	The event occurs periodically, every *X* months.
`eTimerPeriodicYears`	The event occurs periodically, every *X* years.
`eTimerSelective`	The event occurs during selected days of the week.

See Also

`ns_AgentEventGetTimerContext()`,
`ns_AgentEventSetTimerContext()`.

Chapter

8

API Function Reference

ns_AgentActionCompositeParse()

`ns_AgentActionCompositeParse()` retrieves the following data from an action (if specified in the action):

- Email address where the action sends the message
- Return address (the Reply-To header) in the email message
- Contents of the message
- Newsgroup where the action posts the message
- Subject of the message
- Organization in the message header (the Organization header)
- URL to get (via the HTTP GET protocol) and include in the message
- URL and message to post to that URL (via the HTTP POST protocol)
- Query to be sent to a collection
- Name of the collection to send the query to
- Type of search to perform

Call this function if you want to retrieve data from an action (for example, if you are debugging a problem or if you want to copy the data to a new action object).

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_status_t ns_AgentActionCompositeParse(const ns_agentAction_t
myAction,
                             const char** emailTo,
                             const char** retAddr,
                             const char** usrMesg,
                             const char** newsGroup,
                             const char** subject,
                             const char** organization,
                             const char** getUrl,
                             const char** postUrl,
                             const char** postMsg,
                             const char** queryString,
                             const char** collectionName,
                             ESearchQueryType*   searchType);
```

Parameters This function has the following parameters:

`userName`	Name of the user associated with this action.
`userAddr`	Email address of the user associated with this action.
`emailTo`	Email address where the message is sent (the value of the To: field in the message header).
`returnAddr`	Email address where responses should be mailed (the value of the Reply-To field in the mail header).
`mailAndNewsMessage`	Body of the message to be sent by email and/or posted to a newsgroup.
`newsGroup`	Name of the newsgroup where you want to post the message.
`subject`	Subject of the message to be sent or posted (the value of the Subject: field in the mail or news header).
`getFromUrl`	The URL that you want to get by sending an HTTP GET request.
`postToUrl`	The URL that you want to post data to (by sending an HTTP POST request).
`postMessage`	Message that you want to post to the specified URL.
`queryString`	Search criteria that the action should use when searching the collection.
`collectionName`	Name of the collection that you want the action to search.
`searchType`	If `eSearch_All`, returns a list of all matching documents. If `eSearch_by_Mdate`, returns only the documents that have been updated since the agent was last triggered.
`organization`	Name of the organization that you work for (the value appears in the Newsgroups field in the message header).

Returns Returns NS_AGENT_SUCCESS if successful, or NS_AGENT_ACTION_PARSE_ERROR if it could not retrieve data from the action.

See Also

ns_AgentCreateCompositeAction().

ns_AgentActionCreate()

ns_AgentActionCreate() creates a new action and specifies its class. Note that once you create an action, you cannot change its class.

When you are done working with this action, you can free the action from memory by calling the ns_AgentActionDelete() function.

Note that this function allows you to create actions on a more rudimentary level. If you just want to create actions that send mail or post messages to newsgroups, call one of the functions listed under **See Also** instead.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_agentAction_t ns_AgentActionCreate(EAgentActionClass type);
```

Parameters This function has the following parameters:

type	Class of the action that you want to create.

type can be one of the following values, which identifies the class of the action:

eDoNothing	Performs no action.
eCompositeAction	Creates a multipart MIME message out of the results from an HTTP GET, an HTTP POST, and a search. The MIME types of the resulting message parts must be text/html or text/plain. The completed MIME message is either mailed to a specified user or posted to a newsgroup.

Returns Returns the opaque type `ns_agentAction_t`, which represents the newly created action.

Example The following example creates an action of the `eCompositeAction` class.

```
...
#include <agent_c_api.h>
ns_agentAction_t action;
...
action = ns_AgentActionCreate(eCompositeAction);
...
```

See Also

`ns_AgentActionDelete()`.

ns_AgentActionDelete()

`ns_AgentActionDelete()` frees an action from memory. Call this function when you are done working with an action.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
void ns_AgentActionDelete(ns_agentAction_t action);
```

Parameters This function has the following parameters:

`action`	Action that you want to free from memory.

Example The following example frees an action from memory.

```
...
#include <agent_c_api.h>
...
/* Allocate memory for an action */
ns_agentAction_t action = ns_AgentActionCreate();

if (action) {
     ...
     /* Functions for setting the class, context, etc.
        for the action omitted here */
     ...

     /* Create fsan action list */
     actionList = ns_AgentActionListCreate(1);

     /* Add the action to the action list */
     if (actionList)
          ns_AgentActionListAdd(actionList, action);

     /* Or free the action from memory if the
        action list could not be created. */
     else
        ns_AgentActionDelete(action);
     ...
}
...
```

See Also

`ns_AgentActionCreate()`.

ns_AgentActionGetAndMailCreate()

`ns_AgentActionGetAndMailCreate()` creates an action of the `eCompositeAction` class that gets a URL using the HTTP GET method and mails it to a recipient. The function returns the newly created action.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_agentAction_t
ns_AgentActionGetAndMailCreate(const char userName[],
                               const char userAddr[],
                               const char actionEmailAddr[],
                               const char retAddr[],
                               const char getURL[]);
```

Parameters This function has the following parameters:

`userName`	Name of the user associated with the action.
`userAddr`	Email address of the user associated with the action.
`actionEmailAddr`	Email address of the recipient of the message.
`retAddr`	Email address where the recipient should mail responses.
`getURL`	URL to retrieve using the HTTP GET method.

Returns Returns the opaque type `ns_agentAction_t`, which represents the newly created action.

Example The following example gets a URL and mails it to the specified address, along with a message.

```
...
#include <agent_c_api.h>
...
ns_agentAction_t action;
const char *userName = "Mozilla";
const char *userAddress = "mozilla@netscape.com";
const char *mailAddress = "mozilla@netscape.com";
const char *mailMessage = "This is a test message.";
const char *getThisUrl = "http://home.netscape.com/index.html";
...
action = ns_AgentActionGetAndMailCreate(
                            userName,
                            userAddress,
                            mailAddress,
                            mailAddress,
                            mailMessage,
                            getThisUrl);
...
```

See Also

`ns_AgentActionGetAndMailParse()`, `ns_AgentCreateCompositeAction()`.

ns_AgentActionGetAndMailParse()

`ns_AgentActionGetAndMailParse()` retrieves the following data (if specified in the action) from an action that gets a URL and sends it an email message when triggered:

- Email address where the action sends the message
- Return address (the Reply-To header) in the email message
- Contents of the message
- URL to get (via the HTTP GET protocol) and include in the message

Call this function if you want to retrieve data from an action (for example, if you are debugging a problem or if you want to copy the data to a new action object).

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_status_t
ns_AgentActionGetAndMailParse(ns_agentAction_t action,
                              char** actionEmailAddr,
                              char** retAddr,
                              char** getURL);
```

Parameters This function has the following parameters:

`action`	Action that you want to retrieve data from.
`actionEmailAddr`	Email address of the recipient of the message.
`retAddr`	Email address where the recipient should mail responses.
`getURL`	The URL to retrieve using the HTTP GET method.

Returns Returns `NS_AGENT_SUCCESS` if successful, or `NS_AGENT_ACTION_PARSE_ERROR` if it could not retrieve data from the action.

See Also

`ns_AgentActionGetAndMailCreate()`.

ns_AgentActionGetClass()

`ns_AgentActionGetClass()` retrieves the class for a specified action.

Note that you cannot change the class of an action once you have created it.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
EAgentActionClass ns_AgentActionGetClass(const ns_agentAction_t
action);
```

Parameters This function has the following parameters:

`action`	Action that you are getting the class from.

Returns Returns one of the following values, which identifies the class of the action:

`eDoNothing`	Performs no action.
`eCompositeAction`	Creates a multipart MIME message out of the results from an HTTP GET, an HTTP POST, and a search. The MIME types of the resulting message parts must be text/html or text/plain. The completed MIME message is either mailed to a specified user or posted to a newsgroup.

Example The following example gets the class for an action.

```
...
#include <agent_c_api.h>
...
ns_agentAction_t myAction;
EAgentActionClass actionClass;
...
actionClass = ns_AgentActionGetClass(myAction);
...
```

ns_AgentActionGetEMailAddress()

For actions that send an email message to a specified recipient, `ns_AgentActionGetEMailAddress()` retrieves the email address of the recipient.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
char* ns_AgentActionGetEMailAddress(ns_agentAction_t action);
```

Parameters This function has the following parameters:

`action`	Action for which you are getting the recipient's email address.

Returns Returns the email address that the action will send an email message to. If the action is not the type of action that sends an email message, this function returns `NULL`.

(**NOTE:** Do not free the pointer returned by this function call.)

Example The following example gets the email address associated with the specified action.

```
...
#include <agent_c_api.h>
...
ns_agentAction_t action;
char *mailAddress = 0;
...
mailAddress = ns_AgentActionGetEMailAddress(action);
...
```

See Also

`ns_AgentActionGetEMailMessage()`.

ns_AgentActionGetEMailMessage()

For actions that send an email message to a specified recipient, `ns_AgentActionGetEMailMessage()`retrieves the body of the email message that is sent.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
char* ns_AgentActionGetEMailMessage(ns_agentAction_t action);
```

Parameters This function has the following parameters:

`action`	Action for which you are getting the body of the email message sent.

Returns Returns the body of the email message sent when the action is triggered (`NULL` if the email message has an empty body). If the action is not the type of action that sends an email message, this function returns `NULL`.

(**NOTE:** Do not free the pointer returned by this function call.)

Example The following example gets the email message associated with the specified action.

```
...
#include <agent_c_api.h>
...
ns_agentAction_t action;
char *mailMessage = 0;
...
mailMessage = ns_AgentActionGetEMailMessage(ns_agentAction_t action);
...
```

See Also

`ns_AgentActionGetEMailAddress()`.

ns_AgentActionGetUser()

`ns_AgentActionGetUser()` gets information about the user associated with a specified action.

NOTE: Do not free the pointers to the user name and user address.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC

void ns_AgentActionGetUser(ns_agentAction_t action,
                           const char** user,
                           const char** userAddr);
```

Parameters This function has the following parameters:

`action`	Action that you are setting the user for.
`user`	The name of the user associated with the action.
`userAddr`	The email address of the user associated with the action.

Example The following example gets the name and email address of the user associated with the specified action.

```
...
#include <agent_c_api.h>
...
ns_agentAction_t action;
const char *user = 0;
const char *userAddr = 0;
...
ns_AgentActionGetUser(action, &user, &userAddr);
...
```

See Also

`ns_AgentActionSetUser()`.

ns_AgentActionListAdd()

`ns_AgentActionListAdd()` adds an action to a list of actions.

NOTE: Once you add an action object to a list, that list becomes responsible for freeing the action object when the list is freed. If you free the list (by calling the `ns_AgentActionListDelete()` function), you do not need to explicitly free each action object in the list.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
void ns_AgentActionListAdd(const ns_agentActionList_t actionList,
ns_agentAction_t action);
```

Parameters This function has the following parameters:

`actionList`	Action list that you want to add the action to.
`action`	Action that you want to add to the list.

Example The following example adds an action to a list of actions.

```
...
#include <agent_c_api.h>
...
/* Allocate memory for an action */
ns_agentAction_t action = ns_AgentActionCreate();

if (action) {

     ...
     /* Functions for setting the class, context, etc.
        for the action omitted here */
     ...

     /* Allocate memory for an action list */
     actionList = ns_AgentActionListCreate(1);

     /* Add the action to the action list */
     if (actionList)
          ns_AgentActionListAdd(actionList, action);
     ...
}
...
```

See Also

`ns_AgentActionListCount()`, `ns_AgentActionListCreate()`, `ns_AgentActionListDelete()`, `ns_AgentActionListGet()`.

ns_AgentActionListCount()

`ns_AgentActionListCount()` returns the number of actions in a list of actions.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
size_t ns_AgentActionListCount(const ns_agentActionList_t actionList);
```

Parameters This function has the following parameters:

`actionList`	Action list that you want to determine the number of actions in.

Example The following example determines the number of actions in a list of actions and prints out a message if the list is empty.

```
...
#include <stdio.h>
#include <agent_c_api.h>
...
size_t numActions;
ns_agentActionList_t myActionList;
...
numActions = ns_AgentActionListCount(myActionList);
if (numActions == 0) {
    printf("No actions in list.\n");
}
...
```

See Also

`ns_AgentActionListAdd()`, `ns_AgentActionListCreate()`, `ns_AgentActionListDelete()`, `ns_AgentActionListGet()`.

ns_AgentActionListCreate()

`ns_AgentActionListCreate()` creates a new list of actions. Each agent is associated with a list of actions that it invokes when an event occurs.

When you are done working with this list of actions, you can free the list of actions from memory by calling the `ns_AgentActionListDelete()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_agentActionList_t ns_AgentActionListCreate(int numActions);
```

Parameters This function has the following parameters:

`numActions`	Number of actions that the list will contain.

Returns Returns the opaque type `ns_agentActionList_t`, which represents the newly created list of actions.

Example The following example creates a new list of one action and adds an action to that list.

```
...
#include <agent_c_api.h>
...
ns_agentAction_t action;
ns_agentActionList_t actionList;
...
/* Create a new action */
action = ns_AgentActionCreate();

if (action) {

     ...
     /* Functions for setting the class, context, etc.
        for the action omitted here */
     ...

     /* Create a new action list containing one action*/
     actionList = ns_AgentActionListCreate(1);

     /* Add the action to the action list */
     if (actionList)
          ns_AgentActionListAdd(actionList, action);
     ...
}
...
```

See Also

`ns_AgentActionListAdd()`, `ns_AgentActionListCount()`, `ns_AgentActionListDelete()`, `ns_AgentActionListGet()`.

ns_AgentActionListDelete()

`ns_AgentActionListDelete()` frees a list of actions from memory. Call this function when you are done working with a list of actions.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
void ns_AgentActionListDelete(ns_agentActionList_t actionList);
```

Parameters This function has the following parameters:

`actionList`	Action list that you want to free from memory.

Example The following example frees a list of actions from memory after adding the list to an agent.

```
...
#include <agent_c_api.h>
...
ns_status_t status;
```

```
ns_agent_t agent;
ns_agentEvent_t event;
time_t expiryDate;
size_t triggerLimit;
char addr[128], user[128];
ns_agentActionList_t actionList;
...
/* Code for setting the user, address, trigger limit, expiration date,
and event */
...
actionList = ns_AgentActionListCreate(1);
...
/* Code for setting up the list of actions */
...
agent = ns_AgentCreateAgentObject(user, addr, expiryDate, triggerLimit,
                                  0, event, actionList, NULL, 0);
status = ns_AgentSubmit(agent);
ns_AgentActionListDelete(actionList);
ns_AgentDeleteAgentObject(agent);
...
```

See Also

`ns_AgentActionListAdd()`, `ns_AgentActionListCount()`, `ns_AgentActionListCreate()`, `ns_AgentActionListGet()`.

ns_AgentActionListGet()

`ns_AgentActionListGet()` gets an action from a list of actions.

NOTE: Do not free the action object returned by this function call. The action object that you retrieve is still part of the action list. When you free the action list (by calling the `ns_AgentActionListDelete()` function, these action objects will also be freed.

Syntax

```
#include <agent_c_api.h>

...

NSAPI_PUBLIC

ns_agentAction_t ns_AgentActionListGet(const ns_agentActionList_t actionList,
                                                 int index);
```

Parameters This function has the following parameters:

`actionList`	Action list that you want to get the action from.
`index`	The index of the action in the list.

Example The following example gets the first action from a list of actions.

```
...
#include <agent_c_api.h>
...
ns_agentActionList_t myActionList;
size_t numActions;
ns_agentAction_t myAction;
...
numActions = ns_AgentActionListCount(myActionList);
if (numActions  0) {
    myAction = ns_AgentActionListGet(myActionList, 0);
}
...
```

See Also

`ns_AgentActionListAdd()`, `ns_AgentActionListCount()`, `ns_AgentActionListCreate()`, `ns_AgentActionListDelete()`.

ns_AgentActionMailerCreate()

`ns_AgentActionMailerCreate()` creates an action of the `eCompositeAction` class that sends an email message to a designated recipient. The function returns the newly created action.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_agentAction_t
ns_AgentActionMailerCreate(const char userName[],
                           const char userAddr[],
                           const char actionEmailAddr[],
                           const char retAddr[],
                           const char mailContents[]);
```

Parameters This function has the following parameters:

`userName`	Name of the user associated with the action.
`userAddr`	Email address of the user associated with the action.
`actionEmailAddr`	Email address of the recipient of the message.
`retAddr`	Email address where the recipient should mail responses.
`mailContents`	Contents of the email message to send.

Returns Returns the opaque type `ns_agentAction_t`, which represents the newly created action.

Example The following example creates an action of the `eCompositeAction` class that mails a message to a user when invoked.

```
...
#include <agent_c_api.h>
...
ns_agentAction_t action;
char* actionEmailAddr = "user@netscape.com";
char* emailMessage = "The action was invoked.";
char* notifyEmailAddr = "janedoe@netscape.com";
char* userName = "janedoe";
...
action = ns_AgentActionMailerCreate( userName, notifyEmailAddr,
actionEmailAddr, notifyEmailAddr, emailMessage );
...
```

The action defined by this example will email the following message:

```
    From: janedoe@netscape.com
    Reply-To: janedoe@netscape.com
    To: user@netscape.com

    The action was invoked.
```

See Also

`ns_AgentActionMailerParse()`,
`ns_AgentCreateCompositeAction()`.

ns_AgentActionMailerParse()

`ns_AgentActionMailerParse()` retrieves the following data (if specified in the action) from an action that sends an email message when triggered:

- Email address where the action sends the message
- Return address (the Reply-To header) in the email message
- Contents of the message

Call this function if you want to retrieve data from an action (for example, if you are debugging a problem or if you want to copy the data to a new action object).

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_status_t
ns_AgentActionMailerParse(const ns_agentAction_t action,
                          const char** actionEmailAddr,
                          const char** retAddr,
                          const char** emailMsg);
```

Parameters This function has the following parameters:

`action`	Action that you want to retrieve data from.
`actionEmailAddr`	Email address where the message is sent if the agent is triggered.
`retAddr`	Email address where the recipient should mail responses.
`emailMsg`	Body of the email message.

Returns Returns `NS_AGENT_SUCCESS` if successful, or `NS_AGENT_ACTION_PARSE_ERROR` if it could not retrieve data from the action.

See Also

`ns_AgentActionMailerCreate()`.

---

ns_AgentActionNNTPPostCreate()

`ns_AgentActionNNTPPostCreate()` creates an action of the `eCompositeAction` class that posts a message to a newsgroup. The function returns the newly created action.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_agentAction_t
ns_AgentActionNNTPPostCreate(const char userName[],
                             const char userAddr[],
                             const char retAddr[],
                             const char newsGroup[],
                             const char newsSubject[],
                             const char newsMesg[],
                             const char organization[]);
```

Parameters This function has the following parameters:

`userName`	Name of the user associated with the action.
`userAddr`	Email address of the user associated with the action.
`replyAddr`	Email address where the readers should mail responses.
`newsGroup`	Name of the newsgroup where the message will be posted.
`newsSubject`	Subject of the message that will be posted.
`newsMesg`	Body of the message that will be posted.
`organization`	Name of the organization.

Returns Returns the opaque type `ns_agentAction_t`, which represents the newly created action.

Example The following example creates an action that posts a message to a newsgroup.

```
...
#include <agent_c_api.h>
...
ns_agentAction_t action;
const char *userName = "Mozilla";
const char *userAddress = "mozilla@netscape.com";
const char *newsgroup = "alt.test";
const char *subject = "Test of Agents API";
const char *message = "This is a test message.";
const char *organization = "Netscape Communications Corporation";
...
action = ns_AgentActionNNTPPostCreate(
                userName,
                userAddress,
                userAddress,
                newsgroup,
                subject,
                message,
                organization);
...
```

See Also

`ns_AgentActionNNTPPostParse()`, `ns_AgentCreateCompositeAction()`.

ns_AgentActionNNTPPostParse()

`ns_AgentActionNNTPPostParse()` retrieves the following data (if specified in the action) from an action that posts a message to a newsgroup when triggered:

- Newsgroup where the action posts the message
- Subject of the message
- Contents of the message
- Organization in the header of the message (the Organization header)
- Return address (the Reply-To header) in the message

Call this function if you want to retrieve data from an action (for example, if you are debugging a problem or if you want to copy the data to a new action object).

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_status_t
ns_AgentActionNNTPPostParse(const ns_agentAction_t action,
                            const char** newsGroup,
                            const char** newsSubject,
                            const char** newsMesg,
                            const char** organization,
                            const char** replyAddr);
```

Parameters This function has the following parameters:

`action`	Action that you want to retrieve data from.
`newsGroup`	Name of the newsgroup where the message is posted.
`replyAddr`	Email address where readers should mail responses.
`newsSubject`	Subject of the posted message.
`newsMesg`	Body of the posted message.
`organization`	Name of the organization.
`replyAddr`	Email address where readers should mail responses.

Returns Returns `NS_AGENT_SUCCESS` if successful, or `NS_AGENT_ACTION_PARSE_ERROR` if it could not retrieve data from the action.

See Also

`ns_AgentActionNNTPPostCreate()`.

ns_AgentActionPostAndMailCreate()

`ns_AgentActionPostAndMailCreate()` creates an action of the `eCompositeAction` class that posts data to a URL using the HTTP POST method and emails the results to a recipient. The function returns the newly created action.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_agentAction_t
ns_AgentActionPostAndMailCreate(const char userName[],
                                const char userAddr[],
                                const char actionEmailAddr[],
                                const char mailContents[],
                                const char retAddr[],
                                const char postURL[],
                                const char postData);
```

Parameters This function has the following parameters:

`userName`	Name of the user associated with the action.
`userAddr`	Email address of the user associated with the action.
`actionEmailAddr`	Email address of the recipient of the message.
`mailContents`	Contents of the email message to send.
`retAddr`	Email address where the recipient should mail responses.
`postURL`	URL to post data to, using the HTTP POST method.
`postData`	Data to be posted to the specified URL

Returns Returns the opaque type `ns_agentAction_t`, which represents the newly created action.

Example The following example posts data to a URL and mails the result of the post.

```
...
#include <agent_c_api.h>
...
ns_status_t st;
```

```
ns_agent_t agent;
ns_agentEvent_t event;
ns_agentAction_t action;
const char *userName = "Mozilla";
const char *userAddress = "mozilla@netscape.com";
const char *mailAddress = "mozilla@netscape.com";
const char *mailMessage = "This is a test message.";
const char *postToThisUrl = "http://www.netscape.com/myform.html";
const char *postData;
...
agent = ns_AgentActionPostAndMailCreate(userName,
                                        userAddress,
                                        mailAddress,
                                        mailMessage,
                                        mailAddress,
                                        postToThisUrl,
                                        postData);

...
```

See Also

`ns_AgentActionPostAndMailCreate()`, `ns_AgentCreateCompositeAction()`.

ns_AgentActionPostAndMailerParse()

`ns_AgentActionPostAndMailerParse()` retrieves the following data (if specified in the action) from an action that posts a message to a URL and mails the results when triggered:

- Email address where the action sends the message
- Return address (the Reply-To header) in the email message
- URL where you want to post the data to (via the HTTP POST protocol)
- Data that you want to post
- Additional contents to include in the mail message

Call this function if you want to retrieve data from an action (for example, if you are debugging a problem or if you want to copy the data to a new action object).

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_status_t
ns_AgentActionPostAndMailerParse(const ns_agentAction_t action,
                                 const char** actionEmailAddr,
                                 const char** retAddr,
                                 const char** postURL,
                                 const char** postData
                                 const char** usrMesg);
```

Parameters This function has the following parameters:

`action`	Action that you want to retrieve data from.
`actionEmailAddr`	Email address of the recipient of the message.
`retAddr`	Email address where the recipient should mail responses.
`postURL`	The URL to where data is posted, using the HTTP PUT method.
`postData`	The data that is posted to the URL.
`usrMsg`	Body of the email message.

Returns Returns `NS_AGENT_SUCCESS` if successful, or `NS_AGENT_ACTION_PARSE_ERROR` if it could not retrieve data from the action.

See Also

`ns_AgentActionPostMailerCreate()`.

ns_AgentActionSearchCreate()

`ns_AgentActionSearchCreate()` creates an action of the `eCompositeAction` class that searches documents in a specified collection. The action can either mail the results to a user, post the results to a newsgroup, or both. The function returns the newly created action.

Syntax

```
#include <agent_c_api.h>
...
```

```
NSAPI_PUBLIC
ns_agentAction_t
ns_AgentActionSearchCreate(const char userName[],
                           const char userAddr[],
                           const char toAddr[],
                           const char retAddr[],
                           const char usrMesg[],
                           const char newsGroup[],
                           const char newsSubject[],
                           const char organization[],
                           const char replyAddr[],
                           const char collection[],
                           ESearchQueryType searchType,
                           const char searchPattern[]);
```

Parameters This function has the following parameters:

`userName`	Name of the user associated with the action.
`userAddr`	Email address of the user associated with the action.
`toAddr`	Email address where the search results will be sent.
`retAddr`	Email address of the sender.
`usrMesg`	Body of the email message to be sent.
`newsGroup`	Newsgroup where the search results should be posted.
`newsSubject`	Subject of the message to be posted.
`organization`	Name of the organization
`replyAddr`	Email address where replies should be sent.
`collection`	Name of the collection to be searched.
`searchType`	If `eSearch_All`, returns a list of all matching documents. If `eSearch_by_Mdate`, returns only the documents that have been updated since the agent was last triggered.
`searchPattern`	Specifies search criteria (for details on the search syntax, consult the *Netscape Enterprise Server Administrator's Guide*.

Returns Returns the opaque type `ns_agentAction_t`, which represents the newly created action.

Example The following example creates an action that searches a collection.

```
...
#include <agent_c_api.h>
...
ns_agentAction_t action;
const char* user = "mozilla";
const char* mailAddress = "mozilla@netscape.com";
const char* mailMessage = "This is a test.";
const char* collection = "WebPub";
const char* pattern = "(*)";
...
action = ns_AgentActionSearchCreate(user,
                            userAddress,
                            mailAddress,
                            0,
                            mailMessage,
                            0,
                            0,
                            0,
                            0,
                            collection,
                            eSearch_by_Mdate,
                            pattern);
...
```

See Also

`ns_AgentActionSearchParse()`,
`ns_AgentCreateCompositeAction()`.

ns_AgentActionSearchParse()

`ns_AgentActionSearchParse()` retrieves the following data (if specified in the action) from an action that executes a search and mails the results or posts them to a newsgroup when triggered:

- Email address where the action sends the results
- Return address (the Reply-To header) in the email message
- Additional message to include with the results

- Newsgroup where the action posts the results
- Subject of the message
- Organization that appears in the header of the message (the Organization header)
- Collection that you want to search
- Type of search that you want to perform
- Search criteria

Call this function if you want to retrieve data from an action (for example, if you are debugging a problem or if you want to copy the data to a new action object).

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_status_t
ns_AgentActionSearchParse(const ns_agentAction_t myAction,
                          const char** toAddr,
                          const char** retAddr,
                          const char** usrMesg,
                          const char** newsGroup,
                          const char** newsSubject,
                          const char** organization,
                          const char** collection,
                          ESearchQueryType* searchType,
                          const char**searchPattern);
```

Parameters This function has the following parameters:

`toAddr`	Email address where the action sends the message when the event occurs.
`retAddr`	Email address of the sender.
`usrMesg`	Body of the email message that will be sent.
`newsGroup`	Newsgroup where the search results should be posted.
`newsSubject`	Subject of the message to be posted to the newsgroup.
`organization`	Name of the organization

replyAddr	Email address where replies should be sent.
collection	Collection to be searched.
searchType	If eSearch_All, returns a list of all matching documents. If eSearch_by_Mdate, returns only the documents that have been updated since the agent was last triggered.
searchPattern	Specifies search criteria (for details on the search syntax, consult the *Netscape Enterprise Server Administrator's Guide*.

Returns Returns NS_AGENT_SUCCESS if successful, or NS_AGENT_ACTION_PARSE_ERROR if it could not retrieve data from the action.

See Also

ns_AgentActionSearchCreate().

ns_AgentActionSetUser()

ns_AgentActionSetUser() sets the user associated with a specified action.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
void ns_AgentActionSetUser(ns_agentAction_t action, const char user[],
                                 const char userAddr[]);
```

Parameters This function has the following parameters:

action	Action that you are setting the user for.
user	The name of the user that you want associated with the action.
userAddr	The email address of the user that you want associated with the action.

Example The following example sets the user associated with an action.

```
...
#include <agent_c_api.h>
...
ns_agentAction_t action;
const char *user = "Mozilla";
const char *userAddr = "mozilla@netscape.com";
...
ns_AgentActionSetUser(action, user, userAddr);
...
```

See Also

`ns_AgentActionGetUser()`.

---

ns_AgentCreateAgentObject()

`ns_AgentCreateAgentObject()` creates a new agent object.

When you are done working with this agent object, you can free the agent object from memory by calling the `ns_AgentDeleteAgentObject()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_agent_t ns_AgentCreateAgentObject(const char userName[],
                                     const char userAddr[],
                                     time_t expiryDate,
                                     size_t triggerLimit,
                                  const char creationNotificationAddr[],
                                     ns_agentEvent_t adoptEvent,
                                   ns_agentActionList_t adoptActionList,
                                     void* userData,
                                     size_t userDataSize
                                     const char *description);
```

Parameters This function has the following parameters:

`userName`	Name of the user for whom the agent is acting.
`userAddr`	Email address of the user for whom the agent is acting.
`expiryDate`	Date when the agent expires (or 0, if the agent never expires).
`triggerLimit`	Number of times that the actions for this agent can be triggered (when this number is reached, the agent is removed).
`creationNotificationAddr`	Email address to notify when the agent is first created.
`adoptEvent`	Event that the agent should monitor or watch for. This event object is adopted by the agent created. The object will be deleted if the agent object is deleted. (In other words, when you call `ns_AgentDeleteAgentObject()` to delete the agent, this event object will also be deleted.) Do not delete the event object after you pass it to this function.
`adoptActionList`	List of actions that the agent must invoke when the event occurs. This action list object is adopted by the agent created. The object will be deleted if the agent object is deleted. (In other words, when you call `ns_AgentDeleteAgentObject()` to delete the agent, this action list object will also be deleted.) Do not delete the action list object after you pass it to this function.

`userData`	Additional private data that can be used by the agent. Unlike the objects specified in the `adoptEvent` argument and the `adoptActionList` argument, this data is copied into the agent object. The calling function still owns the memory pointed by the `userData` argument and can free that memory, even after the agent object has been created.
`userDataSize`	Length of the additional private data.
`description`	Name of the agent.

Returns Returns the opaque type `ns_agent_t`, which represents the newly created agent object.

Example The following example creates an agent object.

```
...
#include <agent_c_api.h>
...
ns_status_t st;
ns_agent_t agent;
ns_agentEvent_t event;
time_t expiryDate;
size_t triggerLimit;
char addr[128], *user = "myname";
ns_agentActionList_t actionList;
char *desc = "My Agent"
...
/* Specify the maximum number of times that the actions should be
   invoked, the expiration date of the agent, and the email address
   of the user for whom the agent is acting. */
triggerLimit = 50;
expiryDate = time(NULL) + 604800;
sprintf(addr, "%s@mycompany.com", user);

...
/* Set up the event and action list */
...
```

```
/* Create the agent object, and specify values for the
   fields in this object. */
agent = ns_AgentCreateAgentObject(user, addr, expiryDate, triggerLimit,
                                    0, event, actionList, NULL, 0, desc);

/* Register this agent in the Agent Store */
st = ns_AgentSubmit(agent);

/* Now that the agent is registered, you no longer need the agent
   object and can free the object from memory.
   NOTE: This will also free the event object and the action list
   object associated with the agent.  */
ns_AgentDeleteAgentObject(agent);
...
```

See Also

`ns_AgentSubmit()`, `ns_AgentDeleteAgentObject()`.

ns_AgentCreateCompositeAction()

`ns_AgentCreateCompositeAction()` creates an action of the `eCompositeAction` class. You can use actions of this class to send mail messages, post messages to newsgroups, and send requests for URLs (for example, HTTP GET, POST, and PUT requests).

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_agentAction_t ns_AgentCreateCompositeAction(
                        const char userName[],
                        const char userAddr[],
                        const char emailTo[],
                        const char returnAddr[],
                        const char mailAndNewsMessage[],
                        const char newsGroup[],
                        const char subject[],
                        const char getFromUrl[],
                        const char postToUrl[],
                        const char postMessage[],
                        const char queryString[],
                        const char collectionName[],
                        ESearchQueryType   searchType,
                        const char organization[]);
```

Parameters This function has the following parameters:

`userName`	Name of the user associated with this action.
`userAddr`	Email address of the user associated with this action.
`emailTo`	Email address where the message is sent (the value of the To: field in the message header).
`returnAddr`	Email address where responses should be mailed (the value of the Reply-To field in the mail header).
`mailAndNewsMessage`	Body of the message to be sent by email and/or posted to a newsgroup.
`newsGroup`	Name of the newsgroup where you want to post the message.
`subject`	Subject of the message to be sent or posted (the value of the Subject: field in the mail or news header).
`getFromUrl`	The URL that you want to get by sending an HTTP GET request.
`postToUrl`	The URL that you want to post data to (by sending an HTTP POST request).
`postMessage`	Message that you want to post to the specified URL.
`queryString`	Search criteria that the action should use when searching the collection.
`collectionName`	Name of the collection that you want the action to search.
`searchType`	If `eSearch_All`, returns a list of all matching documents. If `eSearch_by_Mdate`, returns only the documents that have been updated since the agent was last triggered.
`organization`	Name of the organization that you work for (the value appears in the Newsgroups field in the message header).

Returns Returns the opaque type `ns_agentAction_t`, which represents the newly created action.

Example The following example creates a new action that does the following:

- Gets a URL and includes the results in a message
- Posts data to a URL
- Sends a mail message
- Posts a message to a newsgroup

```
...
#include <agent_c_api.h>
...
ns_agentAction_t action;
const char* user = "mozilla";
const char* mailAddress = "mozilla@netscape.com";
const char* getThisUrl = "http://home.netscape.com/index.html";
const char* postToUrl = "http://www.netscape.com/myform.html";
const char* postMessage;
const char* mailMessage = "This is a test.";
const char* newsgroup = "alt.test";
const char* subject = "Test from Netscape Agent API";
const char* organization = "ACME, Inc.";
...
action = ns_AgentCreateCompositeAction(
                        user,
                        userAddress,
                        mailAddress,
                        mailAddress,
                        mailMessage,
                        newsgroup,
                        subject,
                        getThisUrl,
                        postToUrl,
                        postMessage,
                        0,
                        0,
                        eSearch_All,
                        organization);
...
```

See Also

`ns_AgentActionCompositeParse()`.

---

ns_AgentDelete()

`ns_AgentDelete()` permanently removes an agent from the Agent Store.

Note
This function permanently removes an agent. Do not call this function if you only intend to free an agent object from memory. To free an agent object, call the `ns_AgentDeleteAgentObject()` function instead.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_status_t ns_AgentDelete(const ns_agentId_t agentId);
```

Parameters This function has the following parameters:

`agentId`	Agent ID identifying the agent that you want to remove from the Agent Store.

Returns Returns `NS_AGENT_SUCCESS` if successful or an error code if unsuccessful.

Example The following example gets a list of the agents owned by the user named "joeuser" and removes the first agent.

```
...
#include <agent_c_api.h>
...
ns_status_t st;
ns_agentId_t *agentListP;
int count;
...
/* Get the list of agents associated with a specific event */
st = ns_AgentGetAgentListByUser("joeuser", &count, &agentListP);

/* Remove the first agent on the list from the Agent Store */
if ((st == NS_AGENT_SUCCESS) && count  0) {
  st = ns_AgentDelete(agentListP[0]);
  /* Free each agent ID in the list (as well as the list itself) from
memory */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentCreateAgentObject()`, `ns_AgentDeleteAgentObject()`, `ns_AgentLookup()`.

ns_AgentDeleteAgentIdList()

`ns_AgentDeleteAgentIdList()` frees the list of the agent IDs and each agent ID allocated by the `ns_AgentGetAgentListByUser()` or `ns_AgentGetAgentListByEvent()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
void ns_AgentDeleteAgentIdList(ns_agentId_t *agentListPP, int count);
```

Parameters This function has the following parameters:

`agentListPP`	Pointer to the list of agent IDs.
`countP`	Number of agent IDs in the list.

Example The following example frees a list of agent IDs and each agent ID in the list.

```
...
#include <agent_c_api.h>
...
int agentCount;
ns_status_t st;
ns_agentId_t *agentListP;

/* Get the list of agent IDs for the agents that work on behalf of ed */
st = ns_AgentGetAgentListByUser("ed", &agentCount, &agentListP);
```

```
if ((st == NS_AGENT_SUCCESS) && count  0) {
  for (int i =0 ; i < agentCount; i++) {

    /* Disable each agent in the list */
    st = ns_AgentDisable(agentListP[i]);
  }

  /* Free each agent ID from memory when done */
  ns_AgentDeleteAgentIdList(agentListP, agentCount);
}
...
```

See Also

`ns_AgentGetAgentListByUser()`, `ns_AgentGetAgentListByEvent()`.

ns_AgentDeleteAgentObject()

`ns_AgentDeleteAgentObject()` frees an agent object from memory. Call this function when you are done working with an agent object created through the `ns_AgentCreateAgentObject()` function or the `ns_AgentLookup()` function.

Note
This function does not permanently remove an agent from the Agent Store; the function simply frees an agent object from memory.

To remove an agent from the Agent Store, call the `ns_AgentDelete()` function instead.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
void ns_AgentDeleteAgentObject(ns_agent_t agent);
```

Parameters This function has the following parameters:

`agent`	Agent object that you want to free from memory.

Example The following example gets a list of the agents for a specific event, gets data from the first agent object, and frees an agent object from memory.

```
...
#include <agent_c_api.h>
...
ns_status_t st;
ns_agentId_t *agentListP;
ns_agent_t agent;
...
/* Get the list of agents owned by joeuser */
st = ns_AgentGetAgentListByUser("joeuser", &count, &agentListP);

/* Get the agent object for the first agent in the list */
if ((st == NS_AGENT_SUCCESS) && count  0) {
    st = ns_AgentLookup(agentListP[0], &agent);
    if (st == NS_AGENT_SUCCESS){

      ...
      /* Get data from the agent object */
      ...

      /* Free the agent object from memory when done */
      ns_AgentDeleteAgentObject(agent);
    }

    /* Free each agent ID in the list, as well as the list itself */
    ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentCreateAgentObject()`, `ns_AgentDelete()`, `ns_AgentLookup()`.

ns_AgentDisable()

`ns_AgentDisable()` disables an agent. If an agent is disabled, the agent is still in the Agent Store, but it does not perform any actions when an event occurs.

If you want to enable a disabled agent, call the `ns_AgentEnable()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_status_t ns_AgentDisable(const ns_agentId_t agentId);
```

Parameters This function has the following parameters:

`agentId`	Agent ID identifying the agent that you want to disable.

Example The following example gets a list of the agents for a specific event and disables the first agent.

```
...
#include <agent_c_api.h>
...
ns_status_t st;
ns_agentId_t *agentListP;
...
/* Get the list of agents owned by "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

/* Remove the first agent on the list from the Agent Store */
if ((st == NS_AGENT_SUCCESS) && count  0) {
  st = ns_AgentDisable(agentListP[0]);

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

ns_AgentEnable().

ns_AgentEnable()

ns_AgentEnable() enables an agent that has been disabled with the ns_AgentDisable() function call.

To determine the state of an agent (for example, to see if the agent is disabled), call the ns_AgentGetState() function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_status_t ns_AgentEnable(const ns_agentId_t agentId);
```

Parameters This function has the following parameters:

agentId	Agent ID identifying the agent that you want to enable.

Returns Returns NS_AGENT_SUCCESS if successful or an error code if unsuccessful.

Example The following example gets a list of the agents for a specific event and enables all disabled agents in that list.

```
...
#include <agent_c_api.h>
...
ns_status_t st;
ns_agentId_t *agentListP;
ns_agent_t agentP;
...
/* Get the list of agents owned by "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

/* Enable any disabled agents */
if ((st == NS_AGENT_SUCCESS) && count  0) {
  for (int i =0 ; i < count; i++) {

    /* Get the agent object for each agent */
    st = ns_AgentLookup(agentListP[i], &agentP);

    /* Check the state of the agent; if disabled, reenable the agent */
    if ( (st == NS_AGENT_SUCCESS) && (ns_AgentGetState(agentP) ==
eDisabled) ) {
        st = ns_AgentEnable(agentListP[i]);
    }
  }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentDisable()`, `ns_AgentGetState()`.

ns_AgentEventCreate()

`ns_AgentEventCreate()` creates a new event.

When you are done working with this event, you can free the event from memory by calling the `ns_AgentEventDelete()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_agentEvent_t ns_AgentEventCreate();
```

Returns Returns the opaque type `ns_agentEvent_t`, which represents the newly created event.

Example The following example creates a new document event.

```
...
#include <agent_c_api.h>
...
int eventId;
char *docUrl = "/myfile.html";
char *docPath = "/usr/ns-home/docs/myfile.html";
ns_agentEvent_t event;
```

```
...
event = ns_AgentEventCreate();
ns_AgentEventSetDocumentContext(event, eventId, docUrl, docPath);
...
```

See Also

`ns_AgentEventDelete()`.

ns_AgentEventDelete()

`ns_AgentEventDelete()` frees an event from memory. Call this function when you are done working with an event.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
void ns_AgentEventDelete(ns_agentEvent_t event);
```

Parameters This function has the following parameters:

`event`	Event that you want to free from memory.

Example The following example frees an event from memory.

```
...
#include <agent_c_api.h>
...
ns_agentEvent_t event;
...
/* Code for setting the user, address, trigger limit, expiration date,
and action list */
...
event = ns_AgentEventCreate();
...
/* Code for setting up the event */
...
agent = ns_AgentCreateAgentObject(user, addr, expiryDate, triggerLimit,
                                  0, event, actionList, NULL, 0, desc);
status = ns_AgentSubmit(agent);
ns_AgentEventDelete(event);
```

```
ns_AgentDeleteAgentObject(agent);
...
```

See Also

`ns_AgentEventCreate()`.

ns_AgentEventGetClass()

`ns_AgentEventGetClass()`retrieves the class for a specified event. You initially specify the class for the event when you create the event.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_agentEventClass_t ns_AgentEventGetClass(const ns_agentEvent_t
event);
```

Parameters This function has the following parameters:

`event`	Event that you are getting the class from.

Returns Returns one of the following values, which identifies the class of the event:

`NS_AGENT_DOCUMENT_EVENT`	The event is a document event (which means that it is based on the state or property of a document).
`NS_AGENT_TIMER_EVENT`	The event is a timer event (which means that it is based on a specific point in time or period of time).

Example The following example uses the `ns_AgentEventGetClass()` function to determine whether to call the `ns_AgentEventGetDocumentContext()` function or the `ns_AgentEventGetTimerContext()` function to get the context of the event.

```
...
#include <agent_c_api.h>
...
ns_agentEvent_t myEvent;
```

```
char* docUrl;
MessageId mesgId;
ns_agent_timer_type tmrType;
time_t *startTime, *prevTrigger, *nextTrigger;
short mnthsVec[12], daysVec[7];
char monthsVecStr[13], daysVecStr[8];
unsigned long period;
...
if (ns_AgentEventGetClass(myEvent) == NS_AGENT_DOCUMENT_EVENT) {
    ns_AgentGetDocumentContext(myEvent, &mesgId, &docUrl);
}
else if (ns_AgentEventGetClass(myEvent) == NS_AGENT_TIMER_EVENT) {
        ns_AgentEventGetTimerContext(myEvent, &tmrType,
                                &startTime, &prevTrigger, &nextTrigger,
                                mnthsVec, daysVec, &period);
}
...
```

See Also

`ns_AgentEventGetDocumentContext()`.

ns_AgentEventGetDocumentContext()

`ns_AgentEventGetDocumentContext()` retrieves the context for a specified document event.

You can set the context for a document event by calling the `ns_AgentEventSetDocumentContext()` function.

Note that the function that calls `ns_AgentEventGetDocumentContext()` is responsible for freeing the pointer to the URL (the `url` argument).

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
int ns_AgentGetDocumentContext(const ns_agentEvent_t event,
                                MessageId* messageId, char** url);
```

Parameters This function has the following parameters:

`event`	Event that you are getting the document context for.
`messageId`	Returns the message ID of the document event.
`url`	Returns the URL of the document.

Returns Returns `NS_AGENT_SUCCESS` if successful or an error code if unsuccessful.

Example The following example retrieves the context from a document event.

```
...
#include <agent_c_api.h>
...
ns_agentEvent_t myEvent;
char* docUrl;
MessageId mesgId;
...
if (ns_AgentEventGetClass(myEvent) == NS_AGENT_DOCUMENT_EVENT) {
    ns_AgentGetDocumentContext(myEvent, &mesgId, &docUrl);
}
...
```

See Also

`ns_AgentEventSetDocumentContext()`.

ns_AgentEventGetMessageId()

`ns_AgentEventGetMessageId()` retrieves the message ID for a specified event.

You can set the message ID for an event by calling the `ns_AgentEventSetMessageId()` function.

Syntax

```
#include <message.h>
...
NSAPI_PUBLIC
MessageId ns_AgentEventGetMessageId(const ns_agentEvent_t event);
```

Parameters This function has the following parameters:

`event`	Event that you are getting the message ID from.

Returns Returns one of the following values, which identifies the message ID of the event (NOTE: this list of events is subject to change):

`NS_DELETE_MSG`	Document was deleted.
`NS_ACCESS_MSG`	Document was accessed.
`NS_PUT_MSG`	Document was modified through the HTTP PUT protocol.
`NS_EDIT_MSG`	Document was edited.
`NS_SAVE_MSG`	Document was saved.
`NS_HEAD_MSG`	Document was accessed through the HTTP HEAD protocol.
`NS_POST_MSG`	Document was accessed through the HTTP POST protocol.
`NS_OPEN_MSG`	Document was opened.
`NS_CLOSE_MSG`	Document was closed.
`NS_READ_MSG`	Document was read.
`NS_WRITE_MSG`	Document was written to.
`NS_MOVE_MSG`	Document was moved.
`NS_QUERY_ATTR_MSG`	Document attributes were queried.
`NS_SET_ATTR_MSG`	Document attributes were set.
`NS_GET_PROPERTIES_MSG`	Document properties were retrieved.
`NS_SET_PROPERTIES_MSG`	Document properties were set.
`NS_COPY_MSG`	Document was copied.
`NS_UNLOCK_MSG`	Document revision was unlocked.
`NS_LOCK_MSG`	Document revision was locked.
`NS_RC_CHECKIN_MSG`	Document revision was checked in.
`NS_RC_CHECKOUT_MSG`	Document revision was checked out.
`NS_RC_GET_LOG_MSG`	Document revision logs were accessed.
`NS_RC_LABEL_MSG`	Document revision was assigned a label.
`NS_RC_GET_REV_MSG`	List of document revisions was accessed.

NS_RC_DIFF_MSG	Document revision was compared ("diff") against another revision.
NS_DELETE_DIR_MSG	Directory was deleted.
NS_MAKE_DIR_MSG	Directory was created.
NS_INDEX_MSG	The contents of the directory were listed.
NS_MOVE_DIR_MSG	Directory was moved.
NS_RM_ALL_DIR_MSG	All directories were removed.
NS_QUERY_DIR_ATTR_MSG	Directory attributes were queried.
NS_SET_DIR_ATTR_MSG	Directory attributes were set.
NS_COPY_DIRMSG	Directory was copied.
NS_TIMER_MSG	Timer event occurred.
NS_VERIFY_LINK	Link report for the document was generated.

Example The following example gets the message ID for an event.

```
...
#include <agent_c_api.h>
...
ns_agentEvent_t myEvent;
MessageId mesgId;
...
mesgId = ns_AgentEventGetMessageId(myEvent);
...
```

See Also

`ns_AgentEventSetMessageId()`.

ns_AgentEventGetMessageIdAsText()

`ns_AgentEventGetMessageIdAsText()` retrieves a text description of the message ID for a specified event.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
```

```
const char* ns_AgentEventGetMessageIdAsText(const ns_agentEvent_t
event);
```

Parameters This function has the following parameters:

`event`	Event that you are getting the text description of the message ID from.

Returns Returns a text description of the message ID associated with the event.

(**NOTE:** Do not free the pointer returned by this function call.)

Example The following example gets the text description of the message ID for an event.

```
...
#include <agent_c_api.h>
...
ns_agentEvent_t myEvent;
...
const char* mesgDesc = ns_AgentEventGetMessageIdAsText(myEvent);
...
```

See Also

`ns_AgentEventGetMessageId()`.

ns_AgentEventGetTimerContext()

`ns_AgentEventGetTimerContext()` retrieves the context for a specified timer event.

You can set the context for a timer event by calling the `ns_AgentEventSetTimerContext()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
int ns_AgentEventGetTimerContext(const ns_agentEvent_t event,
                             ns_agent_timer_type *tmrType,
                             time_t* startTime,
                             time_t* prevTrigger,
time_t* nextTrigger,
short daysVec[7],
unsigned long* period);
```

Parameters This function has the following parameters:

`event`	Timer event that you are getting the context for.
`tmrType`	Returns a value identifying the type of timer used for the event.
`startTime`	Time when the agent was started.
`prevTrigger`	Returns the last point in time when the event occurred.
`nextTrigger`	Returns the next point in time when the event will occur.
`daysVec`	Returns an array of elements representing the days of the week. The elements in the array that are set to 1 correspond to the days of the week when the event will occur. For example, if daysVec[0] = 1, the event activates on Monday. If daysVec[6] = 1, the event activates on Sunday.
`period`	Returns the period of time between events (in seconds).

`tmrType` can have one of the following values:

`eTimerSpecific`	The event occurs at a point in time (for example, June 30, 1997 12:00 PM).
`eTimerPeriodicMinutes`	The event occurs periodically, every *X* minutes (*X* is specified by the `period` parameter).
`eTimerPeriodicHours`	The event occurs periodically, every *X* hours (*X* is specified by the `period` parameter).
`eTimerPeriodicDays`	The event occurs periodically, every *X* days (*X* is specified by the `period` parameter).
`eTimerPeriodicWeeks`	The event occurs periodically, every *X* weeks (*X* is specified by the `period` parameter).
`eTimerPeriodicMonths`	The event occurs periodically, every *X* months (*X* is specified by the `period` parameter).
`eTimerPeriodicYears`	The event occurs periodically, every *X* years (*X* is specified by the `period` parameter).
`eTimerSelective`	The event occurs during selected days of the week (specified by the `daysVec` parameter).

Returns Returns NS_AGENT_SUCCESS if successful or an error code if unsuccessful.

Example The following example .

```
...
#include <agent_c_api.h>
...
ns_status_t st;
ns_agent_t agent;
ns_agentEvent_t event;
ns_agentActionList_t actionList;
...

...
```

See Also

ns_AgentEventSetTimerContext().

ns_AgentEventSetDocumentContext()

ns_AgentEventSetDocumentContext() sets the context for a specified document event.

You can retrieve the context for a document event by calling the ns_AgentEventGetDocumentContext() function.

Syntax

```
#include <agent_c_api.h>

...

NSAPI_PUBLIC

void ns_AgentEventSetDocumentContext(ns_agentEvent_t event,

                                     MessageId messageId,

                                     const char url[], const char pathName[]);
```

Parameters This function has the following parameters:

`event`	Document event that you are setting the context for.
`messageId`	Message ID of the document event.
`url`	URL of the document.
`pathName`	Path of the document.

Example The following example sets the context for the document event.

```
...
#include <agent_c_api.h>
...
int eventId;
char *docUrl = "/myfile.html";
char *docPath = "/usr/ns-home/docs/myfile.html";
ns_agentEvent_t event;
...
event = ns_AgentEventCreate();
ns_AgentEventSetDocumentContext(event, eventId, docUrl, docPath);
...
```

See Also

`ns_AgentEventGetDocumentContext()`.

ns_AgentEventSetMessageId()

`ns_AgentEventSetMessageId()` sets the message ID for a specified event.

You can retrieve the message ID for an event by calling the `ns_AgentEventGetMessageId()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
void ns_AgentEventSetMessageId(ns_agentEvent_t event, MessageId msgId);
```

Parameters This function has the following parameters:

`event`	Event that you are setting the message ID for.
`eventType`	The message ID that you want to set.

`msgId` can be one of the following values, which identifies the message ID of the event (NOTE: this list of events is subject to change):

`NS_DELETE_MSG`	Document was deleted.
`NS_ACCESS_MSG`	Document was accessed.
`NS_PUT_MSG`	Document was modified through the HTTP PUT protocol.
`NS_EDIT_MSG`	Document was edited.
`NS_SAVE_MSG`	Document was saved.
`NS_HEAD_MSG`	Document was accessed through the HTTP HEAD protocol.
`NS_POST_MSG`	Document was accessed through the HTTP POST protocol.
`NS_OPEN_MSG`	Document was opened.
`NS_CLOSE_MSG`	Document was closed.
`NS_READ_MSG`	Document was read.
`NS_WRITE_MSG`	Document was written to.
`NS_MOVE_MSG`	Document was moved.
`NS_QUERY_ATTR_MSG`	Document attributes were queried.
`NS_SET_ATTR_MSG`	Document attributes were set.
`NS_GET_PROPERTIES_MSG`	Document properties were retrieved.
`NS_SET_PROPERTIES_MSG`	Document properties were set.
`NS_COPY_MSG`	Document was copied.
`NS_UNLOCK_MSG`	Document revision was unlocked.
`NS_LOCK_MSG`	Document revision was locked.
`NS_RC_CHECKIN_MSG`	Document revision was checked in.
`NS_RC_CHECKOUT_MSG`	Document revision was checked out.
`NS_RC_GET_LOG_MSG`	Document revision logs were accessed.
`NS_RC_LABEL_MSG`	Document revision was assigned a label.

NS_RC_GET_REV_MSG	List of document revisions was accessed.
NS_RC_DIFF_MSG	Document revision was compared ("diff") against another revision.
NS_DELETE_DIR_MSG	Directory was deleted.
NS_MAKE_DIR_MSG	Directory was created.
NS_INDEX_MSG	The contents of the directory were listed.
NS_MOVE_DIR_MSG	Directory was moved.
NS_RM_ALL_DIR_MSG	All directories were removed.
NS_QUERY_DIR_ATTR_MSG	Directory attributes were queried.
NS_SET_DIR_ATTR_MSG	Directory attributes were set.
NS_COPY_DIRMSG	Directory was copied.
NS_TIMER_MSG	Timer event occurred.
NS_VERIFY_LINK	Link report for the document was generated.

Example The following example sets the message ID for the event to NS_DELETE_MSG, which causes the list of actions to be invoked when the document is deleted.

```
...
#include <agent_c_api.h>
...
ns_agentEvent_t event;
...
ns_AgentEventSetMessageId(event, NS_DELETE_MSG);
...
```

See Also

ns_AgentEventGetMessageId().

ns_AgentEventSetTimerContext()

ns_AgentEventSetTimerContext() sets the context for a specified timer event.

You can retrieve the context for a timer event by calling the ns_AgentEventGetTimerContext() function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
int ns_AgentEventSetTimerContext(ns_agentEvent_t event,
                                 ns_agent_timer_type timer_type,
                                 time_t startTime,
                                 short daysVector[7],
                                 unsigned long period);
```

Parameters This function has the following parameters:

`event`	Timer event that you are setting the context for.
`tmrType`	Value identifying the type of timer used for the event.
`startTime`	Specific point in time when you want the event to occur.
`daysVec`	Array of elements representing the days of the week. The elements in the array that are set to 1 correspond to the days of the week when the event will occur. The array begins with Monday as the first elemnt (index 0) and ends with Sunday as the last element. For example, if daysVec[0] = 1, the event activates on Monday. If daysVec[6] = 1, the event activates on Sunday.
`period`	Period of time between events. Units depend on the timer type selected (for example, for the timer type `eTimerPeriodicMinutes`, `period` is measured in **minutes**).

`tmrType` can have one of the following values:

`eTimerSpecific`	The event occurs at a point in time (for example, June 30, 1997 12:00 PM).
`eTimerPeriodicMinutes`	The event occurs periodically, every *X* minutes (*X* is specified by the `period` parameter).
`eTimerPeriodicHours`	The event occurs periodically, every *X* hours (*X* is specified by the `period` parameter).
`eTimerPeriodicDays`	The event occurs periodically, every *X* days (*X* is specified by the `period` parameter).

`eTimerPeriodicWeeks`	The event occurs periodically, every *X* weeks (*X* is specified by the `period` parameter).
`eTimerPeriodicMonths`	The event occurs periodically, every *X* months (*X* is specified by the `period` parameter).
`eTimerPeriodicYears`	The event occurs periodically, every *X* years (*X* is specified by the `period` parameter).
`eTimerSelective`	The event occurs during selected days of the week (specified by the `daysVec` parameter).

Example The following example .

```
...
#include <agent_c_api.h>
...
ns_status_t st;
ns_agent_t agent;
ns_agentEvent_t event;
ns_agentActionList_t actionList;
...

...
```

See Also

`ns_AgentEventGetTimerContext()`.

ns_AgentGetActionCount()

`ns_AgentGetActionCount()` returns the number of actions in the action list for an agent. To retrieve the actual list of actions, call the `ns_AgentGetActionList()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
size_t ns_AgentGetActionCount(const ns_agent_t agent);
```

Parameters This function has the following parameters:

`agent`	Agent object for which you want to determine the number of actions.

Returns Returns the number of actions in the agent's action list.

Example The following example gets the number of actions associated with an agent.

```
...
#include <agent_c_api.h>
...
size_t numActions;
ns_agent_t agent;
...

/* Get the number of actions in the agent's action list */
numActions = ns_AgentGetActionCount(agent);

...
```

See Also

`ns_AgentGetActionList()`.

---

ns_AgentGetActionList()

`ns_AgentGetActionList()` returns the action list for an agent (the list of actions invoked when a event occurs). To retrieve the number of actions in the list, call the `ns_AgentGetActionCount()` function.

To specify the action list for an agent, call the `ns_AgentSetActionList()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_agentActionList_t ns_AgentGetActionList(const ns_agent_t agent);
```

Parameters This function has the following parameters:

`agent`	Agent object from which you want to retrieve the list of actions.

Returns Returns the opaque type `ns_agentActionList_t`, which represents the list of actions for the agent.

Example The following example gets the list of actions associated with an agent.

```
...
#include <agent_c_api.h>
...
ns_agent_t agent;
ns_actionList_t actionList;
...

/* Get the agent's action list */
actionList = ns_AgentGetActionList(agent);

...
```

See Also

`ns_AgentGetActionCount()`, `ns_AgentSetActionList()`.

ns_AgentGetAgentCount()

`ns_AgentGetAgentCount()` determines the total number of agents in the system.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_status_t ns_AgentGetAgentCount(int *countP);
```

Returns Returns `NS_AGENT_SUCCESS` if successful or an error code if unsuccessful.

Parameters This function has the following parameters:

`countP`	The total number of agents in the system.

Example The following example determines the total number of agents in the system.

```
...
#include <agent_c_api.h>
...
int agentCount;
ns_status_t st;
...

/* Get the total number of agents in the system. */
st = ns_AgentGetAgentCount(&agentCount);

...
```

See Also

`ns_AgentGetUserAgentCount()`.

ns_AgentGetAgentListByEvent()

`ns_AgentGetAgentListByEvent()` gets a list of the agent IDs for the agents that monitor a specified event.

Note that when you are done working with the agent IDs, you must free each agent ID and the list of IDs from memory. To free all agent IDs in the list (as well as the list itself), call the `ns_AgentDeleteAgentIdList()` function.

Syntax

```
#include <agent_c_api.h>

...

NSAPI_PUBLIC

ns_status_t ns_AgentGetAgentListByEvent(const ns_agentEvent_t event,

                                      int *count, ns_agentId_t **agentListPP);
```

Returns Returns `NS_AGENT_SUCCESS` if successful or an error code if unsuccessful.

Parameters This function has the following parameters:

`event`	Event that you want to list the agents for.
`count`	Returns the number of agents that monitor the specified event.
`agentListPP`	Returns an array of agent IDs representing the agents that monitor the specified event.

Example The following example disables all agents for a specified event.

```
...
#include <agent_c_api.h>
...
int agentCount;
ns_status_t st;
ns_agentId_t *agentListP;
ns_agentEvent_t event;
...
/* Get the list of agent IDs for the agents for a specified event */
st = ns_AgentGetAgentListByEvent(event, &agentCount, &agentListP);

if ((st == NS_AGENT_SUCCESS) && count  0) {
  for (int i =0 ; i < agentCount; i++) {

    /* Disable each agent in the list */
    st = ns_AgentDisable(agentListP[i]);

  }

  /* Free the list of agent IDs from memory when done */
  ns_AgentDeleteAgentIdList(agentListP, agentCount);
}
...
```

See Also

`ns_AgentGetUserAgentCount()`, `ns_AgentGetAgentListByUser()`, `ns_AgentDeleteAgentIdList()`.

ns_AgentGetAgentListByUser()

`ns_AgentGetAgentListByUser()` gets a list of the agent IDs for the agents that work on behalf of a particular user. (When you create an agent object by calling the `ns_AgentCreateAgentObject()` function, you specify the name of the user for whom the agent works.)

Note that when you are done working with the agent IDs, you must free each agent ID and the list of IDs from memory. To free all agent IDs in the list (as well as the list itself), call the `ns_AgentDeleteAgentIdList()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_status_t ns_AgentGetAgentListByUser(const char *user, int *countP,
                                      ns_agentId_t **agentListPP);
```

Returns Returns `NS_AGENT_SUCCESS` if successful or an error code if unsuccessful.

Parameters This function has the following parameters:

`user`	User whose agents you want to list.
`countP`	Returns the number of agents that work on behalf of the specified user.
`agentListPP`	Returns an array of agent IDs representing the agents that work on behalf of the specified user.

Example The following example disables all agents for the user named ed.

```
...
#include <agent_c_api.h>
...
int agentCount;
ns_status_t st;
ns_agentId_t *agentListP;

/* Get the list of agent IDs for the agents that work on behalf of ed */
st = ns_AgentGetAgentListByUser("ed", &agentCount, &agentListP);

if ((st == NS_AGENT_SUCCESS) && count  0) {
  for (int i =0 ; i < agentCount; i++) {
    /* Disable each agent in the list */
    st = ns_AgentDisable(agentListP[i]);

  }

  /* Free the list of agent IDs from memory when done */
  ns_AgentDeleteAgentIdList(agentListP, agentCount);
}
...
```

See Also

`ns_AgentGetUserAgentCount()`,
`ns_AgentGetAgentListByEvent()`,
`ns_AgentDeleteAgentIdList()`.

ns_AgentGetCreationDate()

`ns_AgentGetCreationDate()` gets the date when the agent was created.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
time_t ns_AgentGetCreationDate(const ns_agent_t agent);
```

Returns Returns the date when the agent was created.

Parameters This function has the following parameters:

`agent`	Agent object for which you want to get the creation date.

Example The following example gets the creation date for an agent.

```
...
#include <agent_c_api.h>
...
time_t creationDate;
ns_agent_t agent;
ns_status_t st;
ns_agentId_t *agentListP;
```

```
...

/* Get the list of agents owned by "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

if ((st == NS_AGENT_SUCCESS) && count  0) {

  /* Get the first agent from the list of results */
  st = ns_AgentLookup(agentListP[0], &agent);

  if (st == NS_AGENT_SUCCESS) {

    /* Get the agent's creation date */
    creationDate = ns_AgentGetCreationDate(agent);

    /* Free the agent object from memory when done */
    ns_AgentDeleteAgentObject(agent);
  }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentGetLastTriggerDate()`,
`ns_AgentGetLastModifiedDate()`.

ns_AgentGetDescription()

`ns_AgentGetDescription()` gets the name of an agent (this is the name that identifies the agent in the Agent Services page).

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
const char* ns_AgentGetDescription(const ns_agent_t agent);
```

Parameters This function has the following parameters:

`agent`	Agent object whose name you want to get.

Returns Returns the name of the agent. (**NOTE:** Do not free the pointer returned by this function call.)

Example The following example gets the name of an agent.

```
...
#include <agent_c_api.h>
...
const char *agentName;
ns_agent_t agent;
ns_status_t st;
ns_agentId_t *agentListP;
int count;
...

/* Get the list of agents owned by "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

if ((st == NS_AGENT_SUCCESS) && count  0) {

  /* Get the first agent from the list of results */
  st = ns_AgentLookup(agentListP[0], &agent);

  if (st == NS_AGENT_SUCCESS) {

    /* Get the agent's name */
    agentName = ns_AgentGetDescription(agent);

    /* Free the agent object from memory when done */
    ns_AgentDeleteAgentObject(agent);
  }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);

}
...
```

See Also

`ns_AgentSetDescription()`.

---

ns_AgentGetEventInfo()

`ns_AgentGetEventInfo()` returns the event for an agent (the event that causes the agent to invoke its actions). To specify the event for an agent, call the `ns_AgentSetEventInfo()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_agentEvent_t ns_AgentGetEventInfo(const ns_agent_t agent);
```

Parameters This function has the following parameters:

`agent`	Agent object from which you want to retrieve the event.

Returns Returns the opaque type `ns_agentEvent_t`, which represents the event that triggers the agent's actions.

Example The following example gets the event associated with an agent.

```
...
#include <agent_c_api.h>
...
ns_agent_t agent;
ns_agentEvent_t event;
...

/* Get the agent's event */
event = ns_AgentGetEventInfo(agent);

...
```

See Also

`ns_AgentSetEventInfo()`.

ns_AgentGetExpiryDate()

`ns_AgentGetExpiryDate()` gets the expiration date for an agent. If an event triggers the agent after the expiration date, the agent is automatically removed from the Agent Store.

To set the expiration date for an agent, call the `ns_AgentSetExpiryDate()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
time_t ns_AgentGetExpiryDate(const ns_agent_t agent);
```

Parameters This function has the following parameters:

`agent`	Agent object for which you want to get the expiration date.

Returns Returns the expiration date of the agent.

Example The following example returns the expiration date for an agent.

```
...
#include <agent_c_api.h>
...
time_t expiryDate;
ns_agent_t agent;
ns_status_t st;
ns_agentId_t *agentListP;
ns_agentEvent_t event;
...

/* Get the list of agent IDs for janeuser's agents */
st = ns_AgentGetAgentListByUser(&quotjaneuser", &count, &agentListP);

if ((st == NS_AGENT_SUCCESS) && count  0) {

  /* Get the first agent from the list of results */
  st = ns_AgentLookup(agentListP[0], &agent);

  if (st == NS_AGENT_SUCCESS) {
    /* Get the agent's current expiration date */
    expiryDate = ns_AgentGetExpiryDate(agent);

    /* Free the agent object from memory when done */
    ns_AgentDeleteAgentObject(agent);
  }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);

...
```

See Also

`ns_AgentSetExpiryDate()`.

ns_AgentGetId()

`ns_AgentGetId()` retrieves the agent ID for an agent object. You can use this agent ID to find the agent in the Agent Store. (For example, when calling the `ns_AgentLookup()` function to find an agent in the Agent Store, you need to specify the agent ID of the agent you want to find.)

The agent ID is returned in the form of an `ns_agentId_t` opaque type. If you want to get the string representation of this ID, call the `ns_AgentIdToString()` function.

Note that newly created agent objects do not have agent IDs until they are registered with the Agent Store. If you create a new agent object and want to get its agent ID, you must call the `ns_AgentSubmit()` function first to register the agent with the Agent Store before you can call the `ns_AgentGetId()` function to get its agent ID.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
const ns_agentId_t ns_AgentGetId(const ns_agent_t agent);
```

Parameters This function has the following parameters:

`agent`	Agent object for which you want to retrieve the agent ID.

Returns Returns the opaque type `ns_agentId_t`, which represents the agent ID. (**NOTE:** Do not free the pointer returned by this function call.)

Example The following example gets the agent ID for an agent object and converts the ID to a string representation.

```
...
#include <agent_c_api.h>
...
ns_agent_t agent;
ns_agentId_t agentId;
...
/* Get the agent ID for an agent */
agentId = ns_AgentGetId(agent);

/* Convert the ID to a string */
char *stringId = ns_AgentIdToString(agentId);
...
```

See Also

`ns_AgentLookup()`, `ns_AgentDelete()`, `ns_AgentEnable()`, `ns_AgentDisable()`, `ns_AgentIdToString()`, `ns_AgentIdDelete()`.

ns_AgentGetLastModifiedDate()

`ns_AgentGetLastModifiedDate()` gets the last date when the agent was modified.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
time_t ns_AgentGetLastModifiedDate(const ns_agent_t agent);
```

Parameters This function has the following parameters:

`agent`	Agent object for which you want to get the last modified date.

Returns Returns the last date when the agent was modified (or the date when the agent was created, if the agent has never been modified).

Example The following example gets the last modified date for an agent.

```
...
#include <agent_c_api.h>
...
time_t lastModifiedDate;
ns_agent_t agent;
ns_status_t st;
ns_agentId_t *agentListP;
...

/* Get the list of agents owned by "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

if ((st == NS_AGENT_SUCCESS) && count  0) {

  /* Get the first agent from the list of results */
  st = ns_AgentLookup(agentListP[0], &agent);

  if (st == NS_AGENT_SUCCESS) {

    /* Get the agent's last modified date */
    lastModifiedDate = ns_AgentGetLastModifiedDate(agent);

    /* Free the agent object from memory when done */
    ns_AgentDeleteAgentObject(agent);
  }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentGetLastTriggerDate()`, `ns_AgentGetCreationDate()`.

---

ns_AgentGetLastTriggerDate()

`ns_AgentGetLastTriggerDate()` gets the last date when the actions for the agent were triggered (the last date when the event occurred).

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
time_t ns_AgentGetLastTriggerDate(const ns_agent_t agent);
```

Returns Returns the last date when the agent was triggered (or 0, if the agent was never triggered).

Parameters This function has the following parameters:

`agent`	Agent object for which you want to get the last trigger date.

Example The following example gets the last trigger date for an agent.

```
...
#include <agent_c_api.h>
...
time_t lastTriggerDate;
ns_agent_t agent;
ns_status_t st;
ns_agentId_t *agentListP;
...

/* Get the list of agents owned by "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

if ((st == NS_AGENT_SUCCESS) && count  0) {

  /* Get the first agent from the list of results */
  st = ns_AgentLookup(agentListP[0], &agent);

  if (st == NS_AGENT_SUCCESS) {
    /* Get the agent's last trigger date */
    lastTriggerDate = ns_AgentGetLastTriggerDate(agent);

    /* Free the agent object from memory when done */
    ns_AgentDeleteAgentObject(agent);
  }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentSetTriggerLimit()`, `ns_AgentGetTriggerLimit()`, `ns_AgentGetTriggerCount()`.

ns_AgentGetNotificationAddr()

`ns_AgentGetNotificationAddr()` gets the notification address for an agent. This is the email address that will receive a message when the agent is created.

To set the notification address for an agent, call the `ns_AgentSetNotificationAddr()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
const char *ns_AgentGetNotificationAddr(const ns_agent_t agent);
```

Parameters This function has the following parameters:

`agent`	Agent object for which you want to get the notification address.

Returns Returns the notification address for the agent. This is the email address that will receive a message when the agent is created. (**NOTE:** Do not free the pointer returned by this function call.)

Example The following example gets the notification address for an agent.

```
...
#include <agent_c_api.h>
...
size_t triggerLimit;
time_t expiryDate;
ns_agent_t agent;
ns_status_t st;
ns_agentId_t *agentListP;
ns_agentEvent_t event;
char *currentNotifyAddr[255];
char *newNotifyAddr[255];
...

/* Get the list of agents owned by "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

if ((st == NS_AGENT_SUCCESS) && count  0) {

  /* Get the first agent from the list of results */
  st = ns_AgentLookup(agentListP[0], &agent);

  if (st == NS_AGENT_SUCCESS) {

    /* Get the agent's current notification address */
    currentNotifyAddr = ns_AgentGetNotificationAddr(agent);

    /* Free the agent object from memory when done */
    ns_AgentDeleteAgentObject(agent);
  }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentSetNotificationAddr()`.

ns_AgentGetState()

`ns_AgentGetState()` gets the current state of an agent. An agent can be in one of the following states:

- **Enabled.** The agent is in the Agent Store and is active. If the event monitored by the agent occurs, a list of actions associated with the agent is invoked.

  Typically, once you register a new agent with the Agent Store (by calling the `ns_AgentSubmit()` function), the agent is enabled. To reenable a disabled agent, you can call the `ns_AgentEnable()` function).

- **Disabled.** The agent is in the Agent Store but is not active. Even if the event occurs, no actions are invoked.

  You can disable an agent by calling the `ns_AgentDisable()` function.

- **Deleted.** The agent has been marked for removal from the Agent Store. If the event occurs, no actions are invoked.

  You can remove an agent from the Agent Store by calling the `ns_AgentDelete()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
EAgentState ns_AgentGetState(const ns_agent_t agent);
```

Parameters This function has the following parameters:

`agent`	Agent object that you want to get the state of.

Returns Returns one of the following values, which specifies the current state of the agent.

`eEnabled`	The agent is enabled and invokes the specified actions if a particular event occurs.
`eDisabled`	The agent is disabled. Even if the monitored event occurs, none of the actions are invoked.
`eDeleted`	The agent has been marked for removal from the Agent Store.

Example The following example checks the state of janeuser's agents. If any agent is disabled, the example reenables the agent.

```
...
#include <agent_c_api.h>
...
ns_status_t st;
ns_agentId_t *agentListP;
ns_agent_t agentP;
int count, i;
...
/* Get the list of agents owned by the user named "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

/* Enable any disabled agents */
if ((st == NS_AGENT_SUCCESS) && count  0) {
  for (i = 0 ; i < count; i++) {

    /* Get the agent object for each agent */
    st = ns_AgentLookup(agentListP[i], &agentP);

    /* Check the state of the agent; if disabled, reenable the agent */
    if ( (st == NS_AGENT_SUCCESS) && (ns_AgentGetState(agentP) == eDisabled)){
        st = ns_AgentEnable(agentListP[i]);
        /* Free the agent object from memory */
     ns_AgentDeleteAgentObject(agent);
    }

  }

  /* Free each agent ID in the list (and the list itself from memory */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentDisable()`, `ns_AgentEnable()`, `ns_AgentDelete()`.

ns_AgentGetSubType()

`ns_AgentGetSubType()` gets the subtype of an agent. Subtypes are used to distinguish between different agents of the same type. For example, both Timer Agents and Search Agents belong to the Timer Type, even though they behave differently.

In the Agent Services user interface, subtypes are used to distinguish agents apart. An agent can be one of the following subtypes:

- **Document Agent.** This subtype of agent monitors the state and properties of a document.
- **Directory Agent.** This subtype of agent monitors the state and properties of a directory.
- **Timer Agent.** This subtype of agent is invoked at a specific point in time or on a periodic basis.
- **Search Agent.** This subtype of agent executes a search at periodic intervals or at a specific point in time.

Syntax

```
#include <agent_c_api.h>
...
```

```
NSAPI_PUBLIC
eAgentSubType ns_AgentGetSubType(const ns_agent_t agent);
```

Parameters This function has the following parameters:

`agent`	Agent object that you want to get the subtype of.

Returns Returns one of the following values, which specifies the current state of the agent.

`eDocSubType`	Document Agent.
`eDirSubType`	Directory Agent.
`eTimerSubType`	Timer Agent.
`eSearchSubType`	Search Agent.

Example The following example checks the subtype of janeuser's agents and disables any Timer Agents (leaving Search Agents enabled).

```
...
#include <agent_c_api.h>
...
ns_status_t st;
ns_agentId_t *agentListP;
ns_agent_t agentP;
int count, i;
...
/* Get the list of agents owned by the user named "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

/* Disable any Timer Agents */
if ((st == NS_AGENT_SUCCESS) && count  0) {
  for (i = 0 ; i < count; i++) {

    /* Get the agent object for each agent */
    st = ns_AgentLookup(agentListP[i], &agentP);

    /* Check the subtype of the agent; disable it if it is a Timer Agent */
    if ( (st == NS_AGENT_SUCCESS) && (ns_AgentGetSubType(agentP) == eTimerSubType)){
        st = ns_AgentDisable(agentListP[i]);

        /* Free the agent object from memory */
     ns_AgentDeleteAgentObject(agent);
    }

  }

  /* Free each agent ID in the list (and the list itself) from memory */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentSetSubType()`.

ns_AgentGetTriggerCount()

`ns_AgentGetTriggerCount()` gets the current trigger count for an agent. The trigger count is the number of times that the agent has been triggered (the number of times that the event has occurred).

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
size_t ns_AgentGetTriggerCount(const ns_agent_t agent);
```

Returns Returns the current trigger count for the agent.

Parameters This function has the following parameters:

`agent`	Agent object for which you want to get the trigger count.

Example The following example gets the trigger count for an agent.

```
...
#include <agent_c_api.h>
...
size_t triggerCount;
ns_agent_t agent;
ns_status_t st;
ns_agentId_t *agentListP;
...

/* Get the list of agents owned by "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

if ((st == NS_AGENT_SUCCESS) && count  0) {

  /* Get the first agent from the list of results */
  st = ns_AgentLookup(agentListP[0], &agent);

  if (st == NS_AGENT_SUCCESS) {

    /* Get the agent's current trigger count */
    triggerCount = ns_AgentGetTriggerCount(agent);

    /* Free the agent object from memory when done */
    ns_AgentDeleteAgentObject(agent);
  }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentSetTriggerLimit()`, `ns_AgentGetTriggerLimit()`, `ns_AgentGetLastTriggerDate()`.

ns_AgentGetTriggerLimit()

`ns_AgentGetTriggerLimit()` gets the trigger limit for an agent. The trigger limit is the number of times that the agent can be triggered (the number of events that trigger the agent's actions). When an agent has been triggered this number of times, the agent is automatically removed from the Agent Store.

To set the trigger limit for an agent, call the `ns_AgentSetTriggerLimit()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
size_t ns_AgentGetTriggerLimit(const ns_agent_t agent);
```

Returns Returns the trigger limit for the agent.

Parameters This function has the following parameters:

`agent`	Agent object for which you want to get the trigger limit.

Example The following example gets the trigger limit for an agent.

```
...
#include <agent_c_api.h>
...
size_t triggerLimit;
time_t expiryDate;
ns_agent_t agent;
ns_status_t st;
ns_agentId_t *agentListP;
ns_agentEvent_t event;
char *currentNotifyAddr[255];
char *newNotifyAddr[255];
...

/* Get the list of agent IDs for all agents with a particular event */
st = ns_AgentGetAgentListByEvent(event, &count, &agentListP);

if ((st == NS_AGENT_SUCCESS) && count  0) {
  /* Get the first agent from the list of results */
  st = ns_AgentLookup(agentListP[0], &agent);

  if (st == NS_AGENT_SUCCESS) {

    /* Get the agent's current expiration date, and add an hour */
    expiryDate = ns_AgentGetExpiryDate(agent) + 3600;

    /* Get the agent's current trigger limit, and add 5 */
    triggerLimit = ns_AgentGetTriggerLimit(agent) + 5;

    /* Get the agent's current notification address */
    currentNotifyAddr = ns_AgentGetNotificationAddr(agent);

    /* Set a new notification address for the agent */
    ns_AgentSetNotificationAddr(agent, newNotifyAddr);

    /* Set a new expiration date for the agent */
    ns_AgentSetExpiryDate(agent, expiryDate);

    /* Set a new trigger limit for the agent */
    ns_AgentGetTriggerLimit(agent, triggerLimit);

    /* Save the changes in the agent to the Agent Store */
    st = ns_AgentModify(agent);

    /* Free the agent object from memory when done */
    ns_AgentDeleteAgentObject(agent);
  }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentSetTriggerLimit()`, `ns_AgentGetTriggerCount()`, `ns_AgentGetLastTriggerDate()`.

ns_AgentGetUserAgentCount()

`ns_AgentGetUserAgentCount()` determines the number of agents that work on behalf of a particular user. (When you create an agent object by calling the `ns_AgentCreateAgentObject()` function, you specify the name of the user for whom the agent works.)

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_status_t ns_AgentGetUserAgentCount(const char *user, int *countP);
```

Returns Returns `NS_AGENT_SUCCESS` if successful or an error code if unsuccessful.

Parameters This function has the following parameters:

`user`	Name of the user.
`countP`	Number of agents that work for this user.

Example The following example determines the number of agents that work on behalf of a user named admin.

```
...
#include <agent_c_api.h>
...
int agentCount;
ns_status_t st;
...

/* Get the number of agents that work for this user. */
st = ns_AgentGetUserAgentCount("admin", &agentCount);

...
```

See Also

`ns_AgentGetAgentListByUser()`, `ns_AgentCreateAgentObject()`.

ns_AgentGetUserName()

`ns_AgentGetUserName()` gets the name of the user who "owns" the agent. (The agent is working on behalf of this user.) You set this name when creating the agent object with the `ns_AgentCreateAgentObject()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
const char* ns_AgentGetUserName(const ns_agent_t agent);
```

Parameters This function has the following parameters:

`agent`	Agent object for which you want to get the name of the user.

Returns Returns the name of the user for whom the agent is working. (**NOTE:** Do not free the pointer returned by this function call.)

Example The following example gets the user name associated with an agent.

```
...
#include <agent_c_api.h>
...
size_t triggerLimit;
time_t expiryDate;
ns_agent_t agent;
ns_status_t st;
ns_agentId_t *agentListP;
ns_agentEvent_t event;
char *agentUserName[255];
...

/* Get the list of agent IDs for all agents with a particular event */
st = ns_AgentGetAgentListByEvent(event, &count, &agentListP);

if ((st == NS_AGENT_SUCCESS) && count  0) {

  /* Get the first agent from the list of results */
  st = ns_AgentLookup(agentListP[0], &agent);

  if (st == NS_AGENT_SUCCESS) {
    /* Get the name of the user for whom the agent is working */
    agentUserName = ns_AgentGetUserName(agent);

    /* Free the agent object from memory when done */
    ns_AgentDeleteAgentObject(agent);
  }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentCreateAgentObject()`.

ns_AgentIdDelete()

`ns_AgentIdDelete()` frees an agent ID from memory. Call this function when you are done working with an agent ID created through the `ns_AgentGetId()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
void ns_AgentIdDelete(ns_agentId_t agentId);
```

Parameters This function has the following parameters:

agentId	Agent ID that you want to free from memory.

See Also

`ns_AgentGetId()`.

ns_AgentIdFromString()

`ns_AgentIdFromString()` converts the string representation of an agent ID to the `ns_agentId_t` opaque type. Most functions that require the agent ID as a parameter expect the agent ID to be in `ns_agentId_t` opaque type format.

(To get the string representation of an `ns_agentId_t` agent ID, call the `ns_AgentIdToString()` function.)

When you are done working with the agent ID returned by this function, you should free it by calling `ns_AgentIdDelete()`.

(**WARNING:** The string representation of the agent ID cannot have trailing spaces or any other extraneous characters. If you want to verify that the string representation is valid, call the `ns_AgentIdStringIsValid()` function.)

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_agentId_t ns_AgentIdFromString(const char* agentIdStr);
```

Parameters This function has the following parameters:

`agentIdStr`	String representation of the agent ID.

Returns Returns the opaque type `ns_agentId_t`, which represents the agent ID.

Example The following example converts the string representation of an agent ID to the opaque type `ns_agentId_t` and uses this result to get the agent from the Agent Store.

```
...
#include <agent_c_api.h>
...
ns_status_t st;
ns_agent_t agent;
ns_agentId_t agentId;
char *agentIdStr[];
...
/* Get the agent ID from the string representation */
agentId = ns_AgentIdFromString(agentIdStr);
```

```
/* Get the agent with that ID from the Agent Store */
st = ns_AgentLookup(agentId, &agent);
...
```

See Also

`ns_AgentIdToString()`, `ns_AgentIdDelete()`.

ns_AgentIdStringIsValid()

`ns_AgentIdStringIsValid()` checks if the string representation of an agent ID is a valid agent ID.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
int ns_AgentIdStringIsValid(const char* agentIdStr);
```

Parameters This function has the following parameters:

`agentIdStr`	String representation of the agent ID.

Returns Returns 1 if the string representation of the agent ID is valid. If the ID string is invalid, returns 0.

Example The following example verifies that the string representation of an agent ID is valid before converting the string to the opaque type `ns_agentId_t`.

```
...
#include <agent_c_api.h>
...
ns_agentId_t agentId;
char *agentIdStr[];
...
/* Verify that the string representation is valid. */
if (ns_AgentIdStringIsValid(agentIdStr) == 1)  {

/* Get the agent ID from the string representation. */
    agentID = ns_AgentIdFromString(agentIdStr);
}
```

```
...
```

See Also

`ns_AgentIdToString()`, `ns_AgentIdFromString()`.

---

ns_AgentIdToString()

`ns_AgentIdToString()` gets the string representation of an agent ID (in the form of an `ns_agentId_t` opaque type) for an agent object. (To get an agent ID back from its string representation, call the `ns_AgentIdFromString()` function.)

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
const char *ns_AgentIdToString(ns_agentId_t agentId);
```

Parameters This function has the following parameters:

`agent`	Agent ID for which you want the string representation.

Returns Returns the string representation of the agent ID. When you are done working with the agent ID, you need to free it from memory by calling the `ns_AgentIdDelete()` function.

Example The following example gets the agent ID for an agent object and converts the ID to a string representation.

```
...
#include <agent_c_api.h>
...
ns_agent_t agent;
ns_agentId_t agentId;
...
/* Get the agent ID for an agent */
agentId = ns_AgentIdToString(agent);

/* Convert the ID to a string */
char *stringId = ns_AgentIdToString(agentId);
...
```

See Also

ns_AgentIdFromString(), ns_AgentIdDelete().

ns_AgentLookup()

ns_AgentLookup() finds an agent in the Agent Store (using the agent ID), creates an agent object for that agent, and returns the pointer to that agent object.

When you are done working with this agent object, you need to free the agent object from memory by calling the ns_AgentDeleteAgentObject() function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_status_t ns_AgentLookup(const ns_agentId_t agentId, ns_agent_t
*agentP);
```

Parameters This function has the following parameters:

agentId	Agent ID identifying the agent that you want to remove from the Agent Store.
agentP	Returns a pointer to a newly created agent object for this agent.

Returns Returns NS_AGENT_SUCCESS if successful or an error code if unsuccessful.

Example The following example gets an agent based on its agent ID.

```
...
#include <agent_c_api.h>
...
ns_status_t st;
ns_agentId_t agentId;
const char* agentIdAsString = "6|/index.html_1";
...
agentId = ns_AgentIdFromString(agentIdAsString);
st = ns_AgentLookup(agentId);
...
```

See Also

`ns_AgentDeleteAgentObject()`.

ns_AgentModify()

`ns_AgentModify()` saves a modified agent object to the Agent Store.

Note that you cannot change the event associated with an existing, registered agent. If you want to change the event, you need to remove the agent and then create and register it as a new agent in its place.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_status_t ns_AgentModify(const ns_agent_t agent);
```

Parameters This function has the following parameters:

`agent`	The modified agent object that you want to save in the Agent Store.

Returns Returns `NS_AGENT_SUCCESS` if successful or an error code if unsuccessful.

Example The following example creates an agent object for an agent in the Agent Store, changes the trigger limit of the agent, and saves the change back to the Agent Store.

```
...
#include <agent_c_api.h>
...
ns_agentEvent_t event;
ns_status_t st;
ns_agentId_t *agentListP;
ns_agent_t agent;
...

/* Get the list of agents owned by "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

/* Get the agent object for the first agent in the list */
```

```
if ((st == NS_AGENT_SUCCESS) && count  0) {
    st = ns_AgentLookup(agentListP[0], &agent);

    if (st == NS_AGENT_SUCCESS) {

        /* Change the trigger limit to 100 */
        ns_AgentSetTriggerLimit(agent, 100);

        /* Save the change to the Agent Store */
        ns_AgentModify(agent);

        /* Free the agent object from memory when done */
        ns_AgentDeleteAgentObject(agent);
    }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentLookup()`.

---

ns_AgentSetActionList()

`ns_AgentSetActionList()` specifies the action list for an agent (the list of actions invoked when a event occurs). To retrieve the list of actions for an agent, call the `ns_AgentGetActionList()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
void ns_AgentSetActionList(ns_agent_t agent, ns_agentActionList_t
actionList);
```

Parameters This function has the following parameters:

`agent`	Agent object for which you want to set the list of actions.
`actionList`	The list of actions that you want to set. This action list object is adopted by the agent created. The object will be deleted if the agent object is deleted. (In other words, when you call `ns_AgentDeleteAgentObject()` to delete the agent, this action list object will also be deleted.) Do not delete the action list object after you pass it to this function.

Example The following example changes the action list for an agent.

```
...
#include <agent_c_api.h>
...
ns_agentEvent_t event;
ns_status_t st;
ns_agentId_t *agentListP;
ns_agent_t agent;
ns_actionList_t actionList;
int count;
char* actionEmailAddr = "user@netscape.com";
char* emailMesg = "The action was invoked.";
char* notifyEmailAddr = "janedoe@netscape.com";
char* userName = "janedoe";

dataBlock_t mailData;
mailData.data = emailMesg;
mailData.size = strlen(emailMesg) + 1;
...
/* Get the list of agents owned by "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

/* Get the agent object for the first agent in the list */
if ((st == NS_AGENT_SUCCESS) && count  0) {
    st = ns_AgentLookup(agentListP[0], &agent);

    if (st == NS_AGENT_SUCCESS) {
        /* Create a new action list that contains an email action */
        actionList = ns_AgentActionMailerCreate(userName,
notifyEmailAddr,
                                                actionEmailAddr,
notifyEmailAddr,
                                                mailData);

        /* Change the action list for the agent */
        ns_AgentSetActionList(agent, actionList);

        /* Save the change to the Agent Store */
        ns_AgentModify(agent);

        /* Free the agent object from memory when done */
        ns_AgentDeleteAgentObject(agent);
    }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentGetActionList()`, `ns_AgentActionListCreate()`, `ns_AgentActionMailerCreate()`, `ns_AgentActionGetAndMailCreate()`, `ns_AgentActionPostAndMailCreate()`, `ns_AgentActionNNTPPostCreate()`.

ns_AgentSetDescription()

`ns_AgentSetDescription()` sets the name of an agent (this is the name that identifies the agent in the Agent Services page). Note that once an agent has been registered, you cannot change its name. Call this function only for agents that have not been registered yet.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
```

```
void ns_AgentSetDescription(ns_agent_t agent, const char* description);
```

Parameters This function has the following parameters:

`agent`	Agent object whose name you want to set.
`description`	Name that you want to assign to the agent.

Example The following example copies an agent and registers it as a new agent under a different name.

```
...
#include <agent_c_api.h>
...
const char agentName = "My Search Agent";
ns_agent_t agent;
ns_status_t st;
ns_agentId_t *agentListP;
int count;
...

/* Get the list of agent IDs for all agents with a particular event */
st = ns_AgentGetAgentListByEvent(event, &count, &agentListP);

if ((st == NS_AGENT_SUCCESS) && count  0) {

  /* Get the first agent from the list of results */
  st = ns_AgentLookup(agentListP[0], &agent);

  if (st == NS_AGENT_SUCCESS) {

    /* Set the agent's name */
    ns_AgentSetDescription(agent, agentName);

    /* Register this as a new agent */
    ns_AgentSubmit(agent);

    /* Free the agent object from memory when done */
    ns_AgentDeleteAgentObject(agent);
  }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);

}
...
```

See Also

`ns_AgentGetDescription()`.

ns_AgentSetEventInfo()

`ns_AgentSetEventInfo()` sets the event for an agent (the event that causes the agent to invoke its actions). To retrieve the event for an agent, call the `ns_AgentGetEventInfo()` function.

Note that you cannot change the event associated with an existing, registered agent. If you want to change the event, you need to remove the agent and then create and register it as a new agent in its place.

Use this function only with newly created agents that have not been registered.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
void ns_AgentSetEventInfo(ns_agent_t agent, ns_agentEvent_t event);
```

Parameters This function has the following parameters:

`agent`	Agent object for which you want to set the event.
`event`	Event that you want to assign to the agent. This event object is adopted by the agent created. The object will be deleted if the agent object is deleted. (In other words, when you call `ns_AgentDeleteAgentObject()` to delete the agent, this event object will also be deleted.) Do not delete the event object after you pass it to this function.

Example The following example changes the event for an agent.

```
...
```

```
#include <agent_c_api.h>
...
ns_agentEvent_t event;
ns_status_t st;
ns_agentId_t *agentListP;
ns_agent_t agent;
int count;
const char *desc = "My Agent";
...
/* Create a new agent */
agent = ns_AgentCreateAgentObject(user, addr, expiryDate, triggerLimit,
                                  0, NULL, actionList, NULL, 0, desc);

/* Create a new event */
event = ns_AgentEventCreate();

/* Change the event for the agent */
ns_AgentSetEventInfo(agent, event);

/* Register the agent with the Agent Store */
ns_AgentSubmit(agent);
...
```

See Also

`ns_AgentGetEventInfo()`.

ns_AgentSetExpiryDate()

`ns_AgentSetExpiryDate()` sets the expiration date for an agent. If an event triggers the agent after the expiration date, the agent is automatically removed from the Agent Store.

If you specify a date beyond the maximum expiration period allowed by the server administrator, this function changes the expiration date to a date within this period.

To retrieve the expiration date for an agent, call the `ns_AgentGetExpiryDate()` function.

Syntax

```
#include <agent_c_api.h>
```

```
...
NSAPI_PUBLIC
time_t ns_AgentSetExpiryDate(const ns_agent_t agent, time_t
expiryDate);
```

Parameters This function has the following parameters:

`agent`	Agent object for which you want to set the expiration date.
`expiryDate`	The expiration date for the agent.

Returns This function returns the expiration date that was set for the agent. You can use this return value to determine if the server overrode your specified date. (This may happen if you attempt to set a date beyond the maximum expiration period specified by the server administrator.)

Example The following example changes the expiration date for an agent.

```
...
#include <agent_c_api.h>
...
time_t expiryDate;
ns_agent_t agent;
ns_status_t st;
ns_agentId_t *agentListP;
ns_agentEvent_t event;
...

/* Get the list of agents owned by "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

if ((st == NS_AGENT_SUCCESS) && count  0) {

  /* Get the first agent from the list of results */
  st = ns_AgentLookup(agentListP[0], &agent);

  if (st == NS_AGENT_SUCCESS) {

    /* Get the agent's current expiration date, and add an hour */
    expiryDate = ns_AgentGetExpiryDate(agent) + 3600;

    /* Set a new expiration date for the agent */
    ns_AgentSetExpiryDate(agent, expiryDate);
    /* Save the changes in the agent to the Agent Store */
    st = ns_AgentModify(agent);

    /* Free the agent object from memory when done */
    ns_AgentDeleteAgentObject(agent);
  }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentGetExpiryDate()`.

ns_AgentSetNotificationAddr()

`ns_AgentSetNotificationAddr()` sets the notification address for an agent. This is the email address that will receive a message when the agent is created.

To get the notification address for an agent, call the `ns_AgentGetNotificationAddr()` function.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
void ns_AgentSetNotificationAddr(ns_agent_t agent, const char*
address);
```

Parameters This function has the following parameters:

`agent`	Agent object for which you want to set the notification address. This is the email address that will receive a message when the agent is created.
`address`	The notification address for this agent.

Example The following example changes the notification address for an agent.

```
...
#include <agent_c_api.h>
...
ns_agent_t agent;
ns_status_t st;
ns_agentId_t *agentListP;
ns_agentEvent_t event;
char *currentNotifyAddr[255];
char *newNotifyAddr[255];
...

/* Get the list of agents owned by "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

if ((st == NS_AGENT_SUCCESS) && count  0) {

  /* Get the first agent from the list of results */
  st = ns_AgentLookup(agentListP[0], &agent);

  if (st == NS_AGENT_SUCCESS) {

    /* Get the agent's current notification address */
    currentNotifyAddr = ns_AgentGetNotificationAddr(agent);

    /* Set a new notification address for the agent */
    strcpy( newNotifyAddr, "janeuser@mycompany.com" );
    ns_AgentSetNotificationAddr(agent, newNotifyAddr);

    /* Save the changes in the agent to the Agent Store */
    st = ns_AgentModify(agent);

    /* Free the agent object from memory when done */
    ns_AgentDeleteAgentObject(agent);
  }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentGetNotificationAddr()`.

---

ns_AgentSetSubType()

`ns_AgentSetSubType()` sets the subtype of an agent. Subtypes are used to distinguish between different agents of the same type. For example, both Timer Agents and Search Agents belong to the Timer Type, even though they behave differently.

In the Agent Services user interface, subtypes are used to distinguish agents apart. An agent can be one of the following subtypes:

- **Document Agent.** This subtype of agent monitors the state and properties of a document.
- **Directory Agent.** This subtype of agent monitors the state and properties of a directory.
- **Timer Agent.** This subtype of agent is invoked at a specific point in time or on a periodic basis.
- **Search Agent.** This subtype of agent executes a search at periodic intervals or at a specific point in time.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
void ns_AgentSetSubType(ns_agent_t agent, eAgentSubType type);
```

Parameters This function has the following parameters:

`agent`	Agent object that you want to set the subtype of.
`type`	Subtype that you want to set.

Example The following example creates a new agent and assigns it the Search Agent subtype.

```
...
#include <agent_c_api.h>
```

```
...
ns_status_t status;
char* userName = "my_name";
char* notifyEmailAddr = "myname@netscape.com";
char* agentName = "MyAgent";
size_t triggerLimit = 100;
time_t expiryTime = -1;

ns_agent_t myAgent = 0;
ns_agentActionList_t actionList = NULL;
ns_agentEvent_t event = NULL;
ns_agentEventClass_t eventClass;
...
/* Create the new agent. */
myAgent = ns_AgentCreateAgentObject(userName, notifyEmailAddr,
                                    expiryTime, triggerLimit,
                                    notifyEmailAddr,
                                    event, actionList,
                                    NULL, 0,
                                    agentName);

/* Assign the agent a new subtype. */
ns_AgentSetSubType(myAgent, eSearchSubType);
...
```

See Also

`ns_AgentGetSubType()`.

ns_AgentSetTriggerLimit()

`ns_AgentSetTriggerLimit()` sets the trigger limit for an agent. The trigger limit is the number of times that the agent can be triggered (the number of events that trigger the agent's actions). When an agent has been triggered this number of times, the agent is automatically removed from the Agent Store.

To retrieve the trigger limit for an agent, call the `ns_AgentGetTriggerLimit()` function.

Syntax

```
#include <agent_c_api.h>
...
```

```
NSAPI_PUBLIC
int ns_AgentSetTriggerLimit(ns_agent_t agent, int triggerLimit);
```

Parameters This function has the following parameters:

`agent`	Agent object for which you want to set the trigger limit.
`triggerLimit`	The maximum number of times the agent's actions should be triggered.

Example The following example changes the trigger limit for an agent.

```
...
#include <agent_c_api.h>
...
size_t triggerLimit;
ns_agent_t agent;
ns_status_t st;
ns_agentId_t *agentListP;
ns_agentEvent_t event;
...

/* Get the list of agents owned by "janeuser" */
st = ns_AgentGetAgentListByUser("janeuser", &count, &agentListP);

if ((st == NS_AGENT_SUCCESS) && count  0) {

  /* Get the first agent from the list of results */
  st = ns_AgentLookup(agentListP[0], &agent);

  if (st == NS_AGENT_SUCCESS) {

    /* Get the agent's current trigger limit, and add 5 */
    triggerLimit = ns_AgentGetTriggerLimit(agent) + 5;

    /* Set a new trigger limit for the agent */
    ns_AgentSetTriggerLimit(agent, triggerLimit);

    /* Save the changes in the agent to the Agent Store */
    st = ns_AgentModify(agent);

    /* Free the agent object from memory when done */
    ns_AgentDeleteAgentObject(agent);
  }

  /* Free each agent ID and the list of agent IDs when done */
  ns_AgentDeleteAgentIdList(agentListP, count);
}
...
```

See Also

`ns_AgentGetTriggerLimit()`, `ns_AgentGetTriggerCount()`, `ns_AgentGetLastTriggerDate()`.

ns_AgentStatusToString()

`ns_AgentStatusToString()` returns the corresponding error message for an error code of the type `ns_status_t`.

Do not free the string returned by this function. This string is used internally.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
const char *ns_AgentStatusToString(ns_status_t status);
```

Parameters This function has the following parameters:

`status`	Error code returned by an API function.

Example The following example .

```
...
#include <agent_c_api.h>
...
ns_status_t st;
ns_agent_t agent;
ns_agentEvent_t event;
ns_agentActionList_t actionList;
...

...
```

See Also

ns_AgentDisable()

ns_AgentSubmit()

`ns_AgentSubmit()` registers a newly created agent object in the Agent Store.

Syntax

```
#include <agent_c_api.h>
...
NSAPI_PUBLIC
ns_status_t ns_AgentSubmit(ns_agent_t agent);
```

Parameters This function has the following parameters:

`agent`	Agent that you want to register in the Agent Store.

Returns Returns `NS_AGENT_SUCCESS` if successful or an error code if unsuccessful.

Example The following example creates a new agent and registers it in the Agent Store.

```
...
#include <agent_c_api.h>
...
ns_status_t st;
ns_agent_t agent;
ns_agentEvent_t event;
time_t expiryDate;
size_t triggerLimit;
char addr[128], *user = "myname";
ns_agentActionList_t actionList;
char *desc = "My Agent";
...
/* Specify the maximum number of times that the actions should be
   invoked, the expiration date fo the agent, and the email address
   of the user for whom the agent is acting. */
triggerLimit = 50;
expiryDate = time(NULL) + 604800;
sprintf(addr, "%s@mycompany.com", user);
```

```
...
/* Set up the event and action list */
...

/* Allocate memory for the agent object, and specify values for the
   fields in this object. */
agent = ns_AgentCreateAgentObject(user, addr, expiryDate, triggerLimit,
                                  0, event, actionList, NULL, 0, desc);

/* Register this agent in the Agent Store */
st = ns_AgentSubmit(agent);

/* Now that the agent is registered, you no longer need the agent
   object and can free the object from memory. */
ns_AgentDeleteAgentObject(agent);
...
```

See Also

`ns_AgentCreateAgentObject()`.

Chapter 9

Error Code Reference

The following table lists the error codes returned by the agent API functions.

Error Code	**Description**
`NS_AGENT_SUCCESS`	The function successfully completed.
`NS_AGENT_STORE_INIT_FAILED`	The initialization of the Agent Store file failed.
`NS_AGENT_STORE_CORRUPTED`	The Agent Store files are corrupted.
`NS_AGENT_STORE_INCONSISTENT`	The Agent Store files are out of sync.
`NS_AGENT_ENTRY_NOT_FOUND`	The specified entry could not be found in the Agent Store.
`NS_AGENT_ENTRY_ALREADY_EXISTS`	The specified entry already exists in the Agent Store.
`NS_AGENT_CALLER_NOT_AUTHORIZED`	The function caller does not have the required permissions.
`NS_AGENT_USER_LIMIT_REACHED`	The user limit on the number of agents has been reached.
`NS_AGENT_INVALID_EVENT`	The specified event is invalid.
`NS_AGENT_INTERNAL_ERROR`	An internal error occurred in the Agent Subsystem.
`NS_AGENT_ACTION_PARSE_ERROR`	The function could not parse the specified action.