Sun Java System Access Manager 7.1 C API Reference

Beta

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819–4676–05
October 2006

061017@15490

Contents

Preface

The *Sun Java™ System Access Manager 7.1 2006Q4 C API Reference* provides a listing of application programming interfaces (APIs) you can use to enable C applications to access the Sun Java System Access Manager components. The *C API Reference* includes function descriptions and syntax.

Who Should Use This Book

The *C API Reference* is intended for use by IT professionals, network administrators and software developers who implement a network access platform using Sun Java System servers and software. It is recommended that readers of this guide are familiar with the following technologies:

- Lightweight Directory Access Protocol (LDAP)
- Java™
- JavaServer Pages™ (JSP)
- HyperText Transfer Protocol (HTTP)
- HyperText Markup Language (HTML)
- eXtensible Markup Language (XML)
- Web Services Description Language (WSDL)
- Security Assertion Markup Language (SAML)
- SOAP (SOAP is no longer an acronym.)
- Solaris™ Operating System or Linux Application Environment (dependent on deployment platform)
- Web container in which Access Manager will run: Sun Java System Application Server, Sun Java System Web Server, BEA WebLogic, or IBM WebSphere Application Server

Related Books

Access Manager is a component of the Sun Java Enterprise System (Java ES), a software infrastructure that supports enterprise applications distributed across a network or Internet environment. Related documentation is available as follows:

- "Access Manager Core Documentation" on page 14

- "Sun Java Enterprise System Product Documentation" on page 15

Access Manager Core Documentation

The Access Manager core documentation set contains the following titles:

- The *Sun Java System Access Manager 7.1 Release Notes* will be available online after the product is released. It gathers an assortment of last-minute information, including a description of what is new in this current release, known problems and limitations, installation notes, and how to report issues with the software or the documentation.
- The *Sun Java System Access Manager 7.1 Technical Overview* provides an overview of how Access Manager components work together to consolidate access control functions, and to protect enterprise assets and web-based applications. It also explains basic Access Manager concepts and terminology.
- The *Sun Java System Access Manager 7.1 Postinstallation Guide* provides information about specific configurations for Access Manager after installation.
- The *Sun Java System Access Manager 7.1 Deployment Planning Guide* provides information for planning an Access Manager deployment within an existing information technology infrastructure.
- The *Sun Java System Access Manager 7.1 Performance Tuning Guide* provides information on how to tune Access Manager and its related components for optimal performance.
- The *Sun Java System Access Manager 7.1 Administration Guide* describes how to configure, monitor, manage, and maintain Access Manager services, identities, and policies using either the Access Manager Console or the command-line interface.
- The *Sun Java System Access Manager 7.1 2006Q4 Administration Reference* provides reference information for administrators including, for example, error codes.
- The *Sun Java System Access Manager 7.1 Federation and SAML Administration Guide* provides information about the features in Access Manager that are based on the Liberty Alliance Project and SAML specifications. It includes information on the services based on these specifications, instructions for enabling a Liberty-based environment, and summaries of the application programming interface (API) for extending the framework.
- The *Sun Java System Access Manager 7.1 Developer's Guide* offers information on how to customize Access Manager and integrate its functionality into an organization's current technical infrastructure. It also contains details about the programmatic aspects of the product and its API.
- The *Sun Java System Access Manager 7.1 2006Q4 C API Reference* (this guide) provides a listing of APIs you can use to enable C applications to access the Access Manager components. The book includes function descriptions and syntax.
- The *Sun Java System Access Manager 7.1 2006Q4 Java API Reference* is generated from Java™ code using the Javadoc™ tool. The pages provide information on the implementation of the Java packages in Access Manager.
- The *Sun Java System Access Manager Policy Agent 2.2 User's Guide* provides an overview of the policy functionality and the policy agents available for Access Manager.

Updates to the *Sun Java System Access Manager 7.1 Release Notes* and links to modifications of the core documentation can be found on the Access Manager page at the Sun Java Enterprise System documentation web site. Updated documents will be marked with a revision date.

Sun Java Enterprise System Product Documentation

Useful information can be found in the documentation for the following Sun Java System products:

- Directory Server
- Web Server
- Application Server
- Web Proxy Server

Accessing Sun Resources Online

For product downloads, professional services, patches, support, and additional developer information, go to:

- Download Center
- Sun Software Services
- Sun Java Systems Services Suite
- Sun Enterprise Services, Solaris Patches, and Support
- Developer Information

Related Third-Party Web Site References

Third-party URLs are referenced in this documentation set and provide additional, related information. Sun is not responsible for the availability of third-party Web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Feedback

Sun Microsystems is interested in improving its documentation and welcomes your comments and suggestions. To share your thoughts, go to `http://docs.sun.com` and click the Send Comments link at the bottom of the page. In the online form provided, include the document title and part number. The part number is a seven-digit or nine-digit number that can be found on the title page of the book or at the top of the document. For example, the title of this book is *Sun Java System Access Manager 7.1 2006Q2 C API Reference*, and the part number is 819–4676.

Documentation, Support, and Training

Sun Function	URL	Description
Documentation	`http://www.sun.com/documentation/`	Download PDF and HTML documents, and order printed documents
Support and Training	`http://www.sun.com/training/`	Obtain technical support, download patches, and learn about Sun courses

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. Perform a *patch analysis*. Do *not* save the file. [Note that some emphasized items appear bold online.]

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

CHAPTER 1

The C Application Programming Interface Files

Sun Java™ System Access Manager provides C application programming interfaces (APIs) that enable external C applications to participate in Access Manager authentication, authorization, single sign-on (SSO), and logging operations. This chapter covers the following topics:

C Header Files

A *C header file* is a text file that contains pieces of code written in the C programming language. The name of a header file, by convention, ends with the `.h` extension. It is inserted inside a program by coding the `#include` preprocessor directive. By default, Access Manager C header files are installed in the `/`*AccessManager-base*`/SUNWam/include` directory. The Access Manager C header files are:

<am.h>	General utility routines provided by the Access Manager library.
<am_auth.h>	Data types and functions for developing custom authentication modules.
<am_log.h>	Data types and functions for logging on the local system or the Access Manager host.
<am_map.h>	Data types and functions for creating, destroying, and manipulating the map objects used by Access Manager.
<am_notify.h>	Data types and functions for implementing notifications.
<am_policy.h>	Data types and functions for using Access Manager policy objects.
<am_properties.h>	Data types and functions for property maps used by clients of the Access Manager client APIs.
<am_sso.h>	Data types and functions for implementing SSO.
<am_string_set.h>	Data types and functions for manipulating strings.

`<am_types.h>`	Common types and macros provided by Access Manager.
`<am_utils.h>`	This is an unsupported, early access version of utility functions. Functions and data structures may change without backward compatibility.
`<am_web.h>`	Data types and functions intended for use by Access Manager web agents.

C Code Samples

Access Manager provides code samples that demonstrate how you can use the APIs to connect C applications to the Access Manager framework. By default, the code samples are installed in the `/`*AccessManager-base*`/SUNWam/samples/csdk` directory. The code samples are:

`am_auth_test.c` Demonstrates the basic usage of the authentication APIs used to login to an instance of Access Manager.

`am_log_test.c` Demonstrates the basic usage of the logging APIs used to write a message to the Access Manager logs.

`am_policy_test.c` Demonstrates the basic usage of the policy APIs used to evaluate access for specified resources.

Note – Before running the sample, be sure the password defined in the property `com.sun.am.policy.am.password` is in clear text as the sample does not decrypt it. Since `am_policy_test.c` is only a sample, this poses no security risk. For example:

```
com.sun.am.policy.am.username = UrlAccessAgent
com.sun.am.policy.am.password = clear-text-password
```

These properties are defined in `AMAgent.properties`.

`am_sso_test.c` Demonstrates the basic usage of the SSO APIs to perform session operations.

`apache_agent.c` Demonstrates how you can use the policy APIs to build a web agent for the Apache Web Server.

Caution – This is a sample web agent and is not intended to serve as a web agent in a real deployment.

`Makefile` Makefile for building the sample agent.

`README.TXT` Provides detailed instructions for building and executing sample programs.

Required C Libraries

Sample programs are run by launching a generated executable on the command line. The following sections contain instructions for the supported platforms. Be sure to set the library path appropriately for the platform you are using.

- "Solaris Operating System" on page 21
- "Linux Application Environment" on page 21
- "Microsoft® Windows" on page 22

Solaris™ Operating System

Set the `LD_LIBRARY_PATH` environment variable to include the following:

```
/usr/lib/mps:/opt/SUNWam/lib:/usr/lib:/usr/ucblib
```

Note – The `/usr/lib` directory is included before the `/usr/ucblib` directory so that common programs (such as editors) will continue to function.

These directories contain the following shared libraries:

- `libamsdk.so`
- `libxml2.so`
- `libssl3.so`
- `libnss3.so`
- `libplc4.so`
- `libplds4.so`
- `libnspr4.so`
- `libucb.so`

Linux Application Environment

Set the `LD_LIBRARY_PATH` environment variable to include:

/*AccessManager-base*/`agent/lib`

This directory contains the following shared libraries:

- `libamsdk.so`
- `libxml2.so`
- `libssl3.so`
- `libnss3.so`
- `libplc4.so`
- `libplds4.so`

- `libnspr4.so`

Microsoft® Windows

You must have the /*AccessManager-base*/`SUNWam/lib` directory in your path before launching the samples. Alternatively, you can run the `run.bat` script to launch the samples. The script will set your path appropriately.

CHAPTER 2

Authentication Data Types and Functions

Sun Java™ System Access Manager contains public data types and functions you can use in developing custom authentication modules. This chapter provides information and a reference guide to the authentication application programming interface (API). Reference summaries include a short description, syntax, parameters and returns. Prototypes are contained in the `<am_auth.h>` header file located in the `/`*AccessManager-base*`/SUNWam/include` directory. The sample source `am_auth_test.c`, located in the `/`*AccessManager-base*`/SUNWam/samples/csdk` directory, demonstrates the basic usage of the API to login to an instance of Access Manager. This chapter contains the following sections:

The Authentication API for C

C applications can authenticate users with the Access Manager Authentication Service by using the authentication API for C. The C application contacts the Authentication Service to initiate the authentication process, and the Authentication Service responds with a set of requirements. The application then submits authentication credentials back to the Authentication Service and receives further authentication requirements back until there are no more to fulfill. After all requirements have been sent, the client makes one final call to determine if authentication has been successful or has failed.

Authentication Call Sequence

The sequence of calls necessary to authenticate to Access Manager begins with the function call `am_auth_create_auth_context()`. This call returns an `am_auth_context` structure that is then used for the rest of the authentication calls. Once the structure has been initialized, the `am_auth_login()` function is called. This indicates to the Authentication Service that an authentication is desired.

Depending on the parameters passed when creating the `am_auth_context` structure and making the `am_auth_login()` function call, the Authentication Service will determine the login requirements with which to respond. For example, if the requested authentication is to an organization configured for Lightweight Directory Access Protocol (LDAP) authentication with no authentication module chaining involved, the server will respond with a request for a user name and password. The client loops the function call `am_auth_has_more_requirements()`, fills in the needed information and submits this back to the server using the function call `am_auth_submit_requirements()`. (When the requirements are a user name and password, this will happen twice.) The final step is to make the function call `am_auth_get_status()` to determine if the authentication was successful or not.

Note – The `remote-auth.dtd` is the template used to format XML authentication requests sent to Access Manager and to parse XML authentication responses received by the external application. The attributes in the requests/responses correspond to elements in the `remote-auth.dtd`, which can be found in the directory `AccessManager-base/SUNWam/dtd`. In the example, user name corresponds to the `NameCallback` element and password to the `PasswordCallback` element in the `remote-auth.dtd`. More information on `remote-auth.dtd` can be found in Chapter 5, “Using Authentication APIs and SPIs,” in *Sun Java System Access Manager 7.1 Developer's Guide*.

Authentication Properties

The following list of properties are used by the authentication API. Some are defined in the `AMAgent.properties` file and some are not. Those that are not defined can be added to the file so they do not have to be defined for each function call. For example, `com.sun.am.auth.org.name`, which identifies the organization from which you want to authenticate, can be added to `AMAgent.properties`.

Tip – The web agent `AMAgent.properties` includes information for a variety of configurations. By default, the authentication API checks the directory where Access Manager is installed for `AMAgent.properties`. After installing Access Manager though, the file does not exist. If the file does not exist, you must create it and add these properties to the file. More information on `AMAgent.properties` can be found in Appendix C, “Web Agent AMAgent.properties Configuration File,” in *Sun Java System Access Manager Policy Agent 2.2 Guide for Sun Java System Web Server 6.1*.

TABLE 2–1 Properties Needed by the Authentication API for C

Property	Definition
`com.sun.am.naming.url`	URL of the Access Manager Naming Service in the format: **http://***server.domain*:*port***/amserver/namingservice**

TABLE 2–1 Properties Needed by the Authentication API for C *(Continued)*

Property	Definition
`com.sun.am.policy.agents.config.local.log.file`	The logging directory in the format: *path-to-directory*/**`logs/auth-log`** **Note** – This property may be added to `AMAgent.properties`.
`com.sun.am.log.level`	The level at which logs are written in the format: **`all:`***#* where *#* is the level 5 being the highest, 3 medium and 1 the lowest. More information can be found in `AMAgent.properties`.
`com.sun.am.sslcert.dir`	Path to the directory containing the certificate and key databases for Secure Sockets Layer (SSL).
`com.sun.am.certdb.prefix`	Set this property if the certificate databases in the directory specified by **`com.sun.am.sslcert.dir`** has a prefix.
`com.sun.am.certDBPassword=`	The password to the `key3.db` file. **Note** – This property may be added to `AMAgent.properties`.
`com.sun.am.trust_server_certs`	Defines whether or not to trust SSL certificates not defined in the certificate database. Takes a value of `true` or `false` where `true` enables trust.
`com.sun.am.auth.certificateAlias=`	The nick name of the client certificate in the `cert7.db`. **Note** – This property may be added to `AMAgent.properties`.
`com.sun.am.auth.org.name`	The Access Manager organization desired for authentication. The value is the root suffix of the organization using domain-component (dc) as in: **`dc=sun,dc=com`** **Note** – This property may be added to `AMAgent.properties`.

Authentication Data Types

The authentication types defined in `<am_auth.h>` are:

- "`am_auth_context_t`" on page 25
- "`am_auth_callback_t`" on page 26
- "`am_auth_locale_t`" on page 27

am_auth_context_t

Pointer to the authentication context object representing the details of an authentication action.

Syntax

```
#include "am_auth.h"
typedef struct am_auth_context *am_auth_context_t;
```

Members

`am_auth_context` is an opaque structure with no accessible members.

Memory Concerns

The implementation takes care of creating memory.

am_auth_callback_t

Primary callback type for authentication.

Details

`am_auth_callback_t` interacts with the calling application, allowing for the retrieval of specific authentication data (such as user name and password), or the display of error, warning or informational messages. It does not technically retrieve or display the information but provides the means to pass requests between an application and the Access Manager Authentication Service. `struct am_auth_callback` is a C implementation of the Java `javax.security.auth.callback` package. The Java API Reference for this package can be found at `http://java.sun.com/j2se/1.5.0/docs/api/`.

Syntax

```
#include "am_auth.h"
typedef struct am_auth_callback {
    am_auth_callback_type_t callback_type;
    union am_auth_callback_info {
        am_auth_choice_callback_t choice_callback;
        am_auth_confirmation_callback_t confirmation_callback;
        am_auth_language_callback_t language_callback;
        am_auth_name_callback_t name_callback;
        am_auth_password_callback_t password_callback;
        am_auth_text_input_callback_t text_input_callback;
        am_auth_text_output_callback_t text_output_callback;
    } callback_info;
} am_auth_callback_t;
```

Members

`callback_type` — Indicates the kind of callback that will be used. Each `callback_type` has a defined structure with a response field to submit authentication credentials. The value of `callback_type` determines the member of the union defined for the `callback_info` member. The possible values are defined in the enumeration:

```
typedef enum am_auth_callback_type {
    ChoiceCallback = 0,
    ConfirmationCallback,
    LanguageCallback,
    NameCallback,
    PasswordCallback,
    TextInputCallback,
    TextOutputCallback
} am_auth_callback_type_t;
```

Note – Each `callback_type` corresponds to the callback class of the same name in the Java `javax.security.auth.callback` package. The Java API Reference for this package can be found at `http://java.sun.com/j2se/1.5.0/docs/api/`.

`callback_info` — Represents the defined `callback_type`. More information on the specific callbacks can be found in "Authentication Callback Data Types" on page 28.

Memory Concerns

Memory for the callback members is allocated in the `am_auth_login()` call, and freed in the `am_auth_destroy_auth_context()` call. Memory for the response field, though, must be allocated and freed by the caller.

am_auth_locale_t

Data type that holds the attributes that define the locale retrieved in `am_auth_language_callback_t`.

Details

For more information, see "`am_auth_language_callback_t`" on page 30.

Syntax

```
#include "am_auth.h"
typedef struct am_auth_locale {
    const char *language;
```

```
    const char *country;
    const char *variant;
} am_auth_locale_t;
```

Members

`language` Pointer to a valid lower case, two-character ISO—639 language code as defined at `http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt`.

`country` Pointer to a valid upper case, two-character ISO—3166 country code as defined at `http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html`.

`variant` Pointer to a vendor or browser-specific character code. For example, `WIN` for Windows, `MAC` for Macintosh, and `POSIX` for POSIX.

Authentication Callback Data Types

This section contains further details on the callback types as discussed in "`am_auth_callback_t`" on page 26. They are:

- "`am_auth_choice_callback_t`" on page 28
- "`am_auth_confirmation_callback_t`" on page 29
- "`am_auth_language_callback_t`" on page 30
- "`am_auth_name_callback_t`" on page 31
- "`am_auth_password_callback_t`" on page 32
- "`am_auth_text_input_callback_t`" on page 33
- "`am_auth_text_output_callback_t`" on page 33

Note – Each type corresponds to the callback class of the same name in the Java `javax.security.auth.callback` package. The Java API Reference for this package can be found at `http://java.sun.com/j2se/1.5.0/docs/api/`.

`am_auth_choice_callback_t`

Displays a list of choices and submits the selection back to the Access Manager Authentication Service.

Details

`am_auth_choice_callback_t` is a C implementation of the Java `javax.security.auth.callback.ChoiceCallback` class.

Syntax

```
#include "am_auth.h"
typedef struct am_auth_choice_callback {
    const char *prompt;
    boolean_t allow_multiple_selections;
    const char **choices;
    size_t choices_size;
    size_t default_choice;
    const char **response; /* selected indexes */
    size_t response_size;
} am_auth_choice_callback_t;
```

Members

`prompt`	Pointer to the user's prompt.
`allow_multiple_selections`	Takes a value based on the `boolean_t` defined in the `<am_types.h>` header file. Set to `B_TRUE` if multiple selections can be made.
`choices`	Pointer to a pointer to the strings for the different choices.
`choices_size`	Value based on the `size_t` defined in the standard `<stddef.h>` header file that reflects the number of choices in the list.
`default_choice`	Takes a value based on the `size_t` defined in the standard `<stddef.h>` header file that reflects the choice selected by default when the list is displayed.
`response`	Pointer to a pointer to the choice(s) returned to Access Manager.
`response_size`	Takes a value based on the `size_t` defined in the standard `<stddef.h>` header file that reflects the number of selected choices in the response.

Memory Concerns

Memory for the choices list is allocated by `am_auth_login()`, and freed by calling `am_auth_destroy_auth_context()`. Memory for the response must be allocated and freed by the caller.

am_auth_confirmation_callback_t

Requests YES/NO, OK/CANCEL, YES/NO/CANCEL or similar confirmations.

Details

am_auth_confirmation_callback_t is a C implementation of the Java javax.security.auth.callback.ConfirmationCallback class.

Syntax

```
#include "am_auth.h"
typedef struct am_auth_confirmation_callback_info {
    const char *prompt;
    const char *message_type;
    const char *option_type;
    const char **options;
    size_t options_size;
    const char *default_option;
    const char *response; /* selected index */
} am_auth_confirmation_callback_t;
```

Members

prompt	Pointer to the user's prompt.
message_type	Pointer to the message type defined as INFORMATION, WARNING or ERROR.
option_type	Pointer to the option type defined as YES_NO_OPTION, YES_NO_CANCEL_OPTION, OK_CANCEL_OPTION, or UNSPECIFIED.
options	Pointer to a pointer to a list of confirmation options, or NULL if the callback was instantiated with an option_type.
options_size	Takes a value based on the size_t defined in the standard <stddef.h> header file that reflects the number of options in the list.
default_option	Pointer to the option selected by default when the list is displayed.
response	Pointer to the choice returned to Access Manager.

Memory Concerns

Memory is allocated by am_auth_login(), and freed by calling am_auth_destroy_auth_context(). Memory for the response must be allocated and freed by the caller.

am_auth_language_callback_t

Retrieves the locale for localizing text.

Details

`am_auth_language_callback_t` is a C implementation of the Java `javax.security.auth.callback.LanguageCallback` class.

Note – See “am_auth_locale_t” on page 27 for the individual components.

Syntax

```
#include "am_auth.h"
typedef struct am_auth_language_callback_info {
    am_auth_locale_t *locale;
    am_auth_locale_t *response; /* locale */
} am_auth_language_callback_t;
```

Members

`locale`	Pointer to the `am_auth_locale_t` object defining the locale.
`response`	Pointer to the `am_auth_locale_t` object being submitted back to Access Manager.

Memory Concerns

Memory is allocated by `am_auth_login()`, and freed by calling `am_auth_destroy_auth_context()`. Memory for the response must be allocated and freed by the caller.

am_auth_name_callback_t

Retrieves user name information.

Details

`am_auth_name_callback_t` is a C implementation of the Java `javax.security.auth.callback.NameCallback` class.

Syntax

```
#include "am_auth.h"
typedef struct am_auth_name_callback_info {
    const char *prompt;
    const char *default_name;
    const char *response; /* name */
} am_auth_name_callback_t;
```

Members

prompt	Pointer to the user's prompt.
default_name	Pointer to the default name displayed with the user prompt, if any.
response	Pointer to the name submitted back to Access Manager.

Memory Concerns

Memory is allocated by `am_auth_login()`, and freed by calling `am_auth_destroy_auth_context()`. Memory for the response must be allocated and freed by the caller.

am_auth_password_callback_t

Retrieves user password information.

Details

`am_auth_password_callback_t` is a C implementation of the Java `javax.security.auth.callback.PasswordCallback` class.

Syntax

```
#include "am_auth.h"
typedef struct am_auth_password_callback_info {
    const char *prompt;
    boolean_t echo_on;
    const char *response; /* password */
} am_auth_password_callback_t;
```

Members

prompt	Pointer to the user's prompt.
echo_on	Takes a value based on the `boolean_t` defined in the `<am_types.h>` header file. Set to `B_TRUE` to display the password as it is typed.
response	Pointer to the password submitted back to Access Manager.

Memory Concerns

Memory is allocated by `am_auth_login()`, and freed by calling `am_auth_destroy_auth_context()`. Memory for the response must be allocated and freed by the caller.

am_auth_text_input_callback_t

Retrieves generic textual information.

Details

am_auth_text_input_callback_t is a C implementation of the Java javax.security.auth.callback.TextInputCallback class.

Syntax

```
#include "am_auth.h"
typedef struct am_auth_text_input_callback_info {
    const char *prompt;
    const char *default_text;
    const char *response; /* text */
} am_auth_text_input_callback_t;
```

Members

prompt	Pointer to the user's prompt.
default_text	Pointer to the default text to be displayed with the user prompt.
response	Pointer to the text submitted back to Access Manager.

Memory Concerns

Memory is allocated by am_auth_login(), and freed by calling am_auth_destroy_auth_context(). Memory for the response must be allocated and freed by the caller.

am_auth_text_output_callback_t

Displays information messages, warning messages, and error messages.

Details

am_auth_text_output_callback_t is a C implementation of the Java javax.security.auth.callback.TextOutputCallback class.

Syntax

```
#include "am_auth.h"
typedef struct am_auth_text_output_callback_info {
    const char *message;
```

```
    const char *message_type;
} am_auth_text_output_callback_t;
```

Members

message	Pointer to the message to be displayed.
message_type	Pointer to the message type: INFORMATION, WARNING, or ERROR.

Memory Concerns

Memory is allocated by am_auth_login(), and freed by calling am_auth_destroy_auth_context().

Authentication Functions

The authentication functions defined in <am_auth.h> are:

am_auth_abort()

Aborts an authentication process that has not been completed.

Syntax

```
#include "am_auth.h"
AM_EXPORT am_status_t
am_auth_abort(am_auth_context_t auth_ctx);
```

Parameters

This function takes the following parameter:

`auth_ctx` The `am_auth_context_t` type.

Note – See "`am_auth_context_t`" on page 25 for information.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the <am_types.h> header file):

`AM_SUCCESS`	If the process was successfully stopped.
`AM_INVALID_ARGUMENT`	If the `auth_ctx` parameter is `NULL`.

am_auth_create_auth_context()

Creates the context for the authentication and a pointer to it.

Syntax

```
#include "am_auth.h"
AM_EXPORT am_status_t
am_auth_create_auth_context(am_auth_context_t *auth_ctx,
                            const char *org_name,
                            const char *cert_nick_name,
                            const char *url);
```

Parameters

This function takes the following parameters:

`auth_ctx` Pointer to the `am_auth_context_t` type.

Note – See "`am_auth_context_t`" on page 25 for information.

`org_name`	Pointer to the name of the organization for which the authentication context is being initialized. May be `NULL` to use the value defined in the `AMAgent.properties` file.
`cert_nick_name`	Pointer to the alias of the certificate being used if the application will connect securely to Access Manager. May be `NULL` if the connection is not secure.
`url`	Pointer to the Access Manager Naming Service URL. May be `NULL` to use the Naming Service URL defined in the `AMAgent.properties` file.

Returns

This function returns a pointer to the authentication context object and one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the authentication context was successfully created.
AM_NO_MEMORY	If unable to allocate memory for the handle.
AM_INVALID_ARGUMENT	If the auth_ctx parameter is NULL.
AM_AUTH_CTX_INIT_FAILURE	If the authentication initialization failed.

am_auth_destroy_auth_context()

Eliminates the specified authentication context.

Syntax

```
#include "am_auth.h"
AM_EXPORT am_status_t
am_auth_destroy_auth_context(am_auth_context_t auth_ctx);
```

Parameters

This function takes the following parameter:

auth_ctx The am_auth_context_t type.

Note – See "am_auth_context_t" on page 25 for information.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the pointer was successfully destroyed.
AM_INVALID_ARGUMENT	If the auth_ctx parameter is NULL.

am_auth_get_callback()

Retrieves the appropriate callback structure to populate with authentication requirements.

Syntax

```
#include "am_auth.h"
AM_EXPORT am_auth_callback_t *
am_auth_get_callback(am_auth_context_t auth_ctx,
                     size_t index);
```

Parameters

This function takes the following parameters:

auth_ctx The am_auth_context_t type.

Note – See "am_auth_context_t" on page 25 for information.

index Takes a value based on size_t defined in the standard <stddef.h> header file that initializes the index into the callback array.

Returns

This function returns a pointer to the am_auth_callback_t type. See "am_auth_callback_t" on page 26 for more information.

am_auth_get_module_instance_names()

Retrieves the authentication module plug-in instances configured for the organization (or sub-organization) defined in the am_auth_context_t type.

Details

Module instance names are retrieved in pointer to a pointer to a am_string_set_t type (as defined in the <am_string_set.h> header file).

Syntax

```
#include "am_auth.h"
AM_EXPORT am_status_t
am_auth_get_module_instance_names(am_auth_context_t auth_ctx,
                                  am_string_set_t** module_inst_names_ptr);
```

Parameters

This function takes the following parameters:

auth_ctx The am_auth_context_t type.

Note – See "am_auth_context_t" on page 25 for information.

`module_inst_names_ptr`	Pointer to a pointer to the `am_string_set_t` type.

Returns

This function returns a pointer to a pointer with the list of module instance names (or `NULL` if the number of configured modules is zero) and one of the following values of the `am_status_t` enumeration (defined in the <am_types.h> header file):

`AM_SUCCESS`	If the submitted requirements were processed successfully.
`AM_AUTH_FAILURE`	If the authentication process failed.
`AM_INVALID_ARGUMENT`	If the `auth_ctx` parameter is `NULL`.
`AM_SERVICE_NOT_INITIALIZED`	If the Access Manager Authentication Service is not initialized.

Memory Concerns

The implementation takes care of allocating memory for the `module_inst_names_ptr`.

am_auth_get_organization_name()

Retrieves the organization to which the user is authenticated.

Syntax

```
#include "am_auth.h"
AM_EXPORT const char *
am_auth_get_organization_name(am_auth_context_t auth_ctx);
```

Parameters

This function takes the following parameter:

`auth_ctx`	The `am_auth_context_t` type.

Note – See "am_auth_context_t" on page 25 for information.

Returns

This function returns a pointer with one of the following values:

Zero-terminated string representing the organization.
After the user successfully logs in.

`NULL`
If there was an error or the user has not successfully logged in.

am_auth_get_sso_token_id()

Retrieves the session identifier for the authenticated user.

Details

The *SSOTokenID* is a randomly-generated string that represents an authenticated user. See "Single Sign-on Token Handles" on page 68 for more information.

Syntax

```
#include "am_auth.h"
AM_EXPORT const char *
am_auth_get_sso_token_id(am_auth_context_t auth_ctx);
```

Parameters

This function takes the following parameter:

`auth_ctx` The `am_auth_context_t` type.

Note – See "`am_auth_context_t`" on page 25 for information.

Returns

This function returns a pointer with one of the following values:

Zero-terminated string representing the session token.
After the user successfully logs in.

`NULL`
If there was an error or the user has not successfully logged in.

am_auth_get_status()

Retrieves the state of the authentication process.

Syntax

```
#include "am_auth.h"
AM_EXPORT am_auth_status_t
am_auth_get_status(am_auth_context_t auth_ctx);
```

Parameters

This function takes the following parameter:

`auth_ctx`	The `am_auth_context_t` type.

Note – See "`am_auth_context_t`" on page 25 for information.

Returns

This function returns one of the following values of the `am_auth_status_t` enumeration as defined:

```
typedef enum am_auth_status {
    AM_AUTH_STATUS_SUCCESS = 0,
    AM_AUTH_STATUS_FAILED,
    AM_AUTH_STATUS_NOT_STARTED,
    AM_AUTH_STATUS_IN_PROGRESS,
    AM_AUTH_STATUS_COMPLETED
} am_auth_status_t;
```

`AM_AUTH_STATUS_FAILED`	The login process has failed.
`AM_AUTH_STATUS_NOT_STARTED`	The login process has not started.
`AM_AUTH_STATUS_IN_PROGRESS`	The login is in progress.
`AM_AUTH_STATUS_COMPLETED`	The user has been logged out.
`AM_AUTH_STATUS_SUCCESS`	The user has logged in.

`am_auth_has_more_requirements()`

Checks to see if there are additional requirements needed to complete the login process.

Details

`am_auth_has_more_requirements()` is invoked after the `am_auth_login()` call. If there are requirements to be supplied, the caller retrieves and submits the requirements in the form of callbacks.

Syntax

```
#include "am_auth.h"
AM_EXPORT boolean_t
am_auth_has_more_requirements(am_auth_context_t auth_ctx);
```

Parameters

This function takes the following parameter:

`auth_ctx` The `am_auth_context_t` type.

Note – See “`am_auth_context_t`” on page 25 for information.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the `<am_types.h>` header file):

`B_TRUE` If there are more requirements.

`B_FALSE` If there are no more requirements.

`am_auth_init()`

Initializes the authentication module using the pointer returned by `am_auth_create_auth_context()`.

Syntax

```
#include "am_auth.h"
AM_EXPORT am_status_t
am_auth_init(const am_properties_t auth_init_params);
```

Parameters

This function takes the following parameter:

`auth_init_params` The `am_properties_t` type which contains the module initialization properties.

Note – See “`am_properties_t`” on page 131 for information.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS` If the initialization of the library is successful.

`AM_NO_MEMORY` If unable to allocate memory during initialization.

AM_INVALID_ARGUMENT	If auth_init_params is NULL.
Others	See “am_status_t” on page 209.

am_auth_login()

Begins the login process given the index type and its value.

Syntax

```
#include "am_auth.h"
AM_EXPORT am_status_t
am_auth_login(am_auth_context_t auth_ctx,
              am_auth_index_t auth_idx,
              const char *value);
```

Parameters

This function takes the following parameters:

auth_ctx — The am_auth_context_t type.

> **Note –** See “am_auth_context_t” on page 25 for information.

auth_idx — Defines the resource for which the authentication is being performed. Based on the am_auth_index_t enumeration used to initiate the login process:

```
typedef enum am_auth_idx {
    AM_AUTH_INDEX_AUTH_LEVEL = 0,
    AM_AUTH_INDEX_ROLE,
    AM_AUTH_INDEX_USER,
    AM_AUTH_INDEX_MODULE_INSTANCE,
    AM_AUTH_INDEX_SERVICE
} am_auth_index_t;
```

value — Pointer to the authentication module being used.

> **Note –** See “Authentication Module Types” in *Sun Java System Access Manager 7.1 Administration Guide* for more information.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the login process was successfully completed.
AM_INVALID_ARGUMENT	If the auth_ctx or value parameter is NULL.
AM_FEATURE_UNSUPPORTED	If the auth_idx parameter is invalid.

am_auth_logout()

Logs out the user.

Syntax

```
#include "am_auth.h"
AM_EXPORT am_status_t
am_auth_logout(am_auth_context_t auth_ctx);
```

Parameters

This function takes the following parameter:

auth_ctx The am_auth_context_t type.

Note – See “am_auth_context_t” on page 25 for information.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the logout process was successfully completed.
AM_INVALID_ARGUMENT	If the auth_ctx parameter is NULL.

am_auth_num_callbacks()

Retrieves the number of callbacks.

Syntax

```
#include "am_auth.h"
AM_EXPORT size_t
am_auth_num_callbacks(am_auth_context_t auth_ctx);
```

Parameters

This function takes the following parameters:

`auth_ctx`	The `am_auth_context_t` type.

Note – See "`am_auth_context_t`" on page 25 for information.

Returns

This function returns a value based on the `size_t` defined in the standard `<stddef.h>` header file that reflects the number of callbacks.

`am_auth_submit_requirements()`

Passes the responses populated in the callbacks to the Authentication Service.

Syntax

```
#include "am_auth.h"
AM_EXPORT am_status_t
am_auth_submit_requirements(am_auth_context_t auth_ctx);
```

Parameters

This function takes the following parameter:

`auth_ctx`	The `am_auth_context_t` type.

Note – See "`am_auth_context_t`" on page 25 for information.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the submitted requirements were processed successfully.
`AM_AUTH_FAILURE`	If the authentication process failed.
`AM_INVALID_ARGUMENT`	If the `auth_ctx` parameter is `NULL`.

CHAPTER 3

Policy Data Types and Functions

Sun™ Java System Access Manager contains public data types and functions you can use to communicate with the Policy Service. Reference summaries include a short description, syntax, parameters and returns. The code is contained in the `<am_policy.h>` header file (located in the /*AccessManager-base*/`SUNWam/include` directory). The sample source `am_policy_test.c` (located in the /*AccessManager-base*/`SUNWam/samples/csdk` directory) demonstrates the basic usage of the policy API. This chapter contains the following sections:

- "The Policy API for C" on page 45
- "Policy Data Types" on page 47
- "Policy Functions" on page 51

The Policy API for C

Access Manager provides policy APIs for use by developers to integrate a resource authorization functionality within their external C applications. The policy API for C determines if a user has been given permission by a recognized authority to access a particular protected resource. The result of the policy evaluation is called an *action value* and may be boolean or binary.

- A *boolean action value* might be allow/deny or yes/no.
- A *binary action value* might be, for example, a mailbox quota. Assuming John Smith can only hold 100 MB of email in his mailbox, the value 100 would be the action value.

Tip – As policy evaluation results in string values only, the policy evaluation returned is 100 numeric and not 100 MB. It is up to the application developer to define the appropriate metric for the values.

Resources Strings

The Policy API for C mandates that any resource be represented in a string format. Thus, resources on a web server must be represented as URLs. The Policy Service is then able to compare the resource string to the policy string and determine a *relative relationship* between the two. This relationship will be defined as one of the following:

- exact match
- no match
- subordinate match
- superior match
- exact pattern match

Note – Exact pattern match is a special case where resources may be represented collectively as patterns. The information is abstracted from the Policy Service and the comparison operation must take a boolean parameter to trigger a pattern matched comparison. During the caching of policy information, the policy engine does not care about patterns, whereas during policy evaluation, the comparisons are pattern sensitive.

Resource Traits

The set of characteristics needed to define a resource is called a *resource trait*. Resource traits are taken as a parameter during service initialization in the "`am_resource_traits_t`" on page 49. Using the resource traits, the Policy Service constructs a resource graph for policy evaluation in which the relation between all resources in the system spans out like a tree from the root of the given resource. Thus, the service developer must provide the means to extract the root of the given resource. In a URL, the *protocol*`://`*Access Manager-host*`.`*domain*`:`*port* portion represents the root.

Policy Evaluation

The two `typedef` structures that are used for information exchange to and from the policy evaluation interfaces are:

- `am_map_t` provides a key to multiple key/value mapping. If the evaluation requires certain environment parameters like the IP address of the requester, it may be passed using this structure. See "`am_map_t`" on page 113 for more information.
- `am_properties_t` provides a key to single value mapping. `am_properties_t` provides the additional functionality of loading a configuration file and getting values of specific data types. See "`am_properties_t`" on page 131 for more information.

Policy Data Types

The types defined in <am_policy.h> are:

- "am_policy_result_t" on page 47
- "am_policy_t" on page 49
- "am_resource_traits_t" on page 49

am_policy_result_t

Carries the evaluation results from the Policy Service.

Details

am_policy_result_t unifies various components of a policy evaluation including information regarding the user attempting to perform an action on the resource, *advice messages* as recommended during policy evaluation, if any, and attribute response maps providing specific key/values as set in policy definition or user entries.

Syntax

```
#include "am_policy.h"
typedef struct am_policy_result {
    const char *remote_user;
    const char *remote_user_passwd;
    const char *remote_IP;
    am_map_t advice_map;
    am_map_t attr_profile_map;
    am_map_t attr_session_map;
    am_map_t attr_response_map;
    const char *advice_string;
} am_policy_result_t;
```

Members

remote_user	Pointer to the user attempting access.
remote_user_passwd	Pointer to the password for the remote user.
remote_IP	Pointer to the IP address of the resource the user is attempting to access.
advice_map	Takes a value based on the am_map_t defined in the <am_map.h> header file that represents any *advice messages* that might have resulted from the policy evaluation.

Note – For information on advices, see "Policy Advices" in *Sun Java System Access Manager 7.1 Administration Guide*.

`attr_profile_map`
: Takes a value based on the `am_map_t` (defined in the `<am_map.h>` header file) that represents one or more user profile attributes and a corresponding value. This member is enabled when the following two properties in `AMAgent.properties` are configured:

- `com.sun.am.policy.agents.config.profile.attribute.fetch.mode` takes a value of `HTTP_HEADER` or `HTTP_COOKIE`.
- `com.sun.am.policy.agents.config.profile.attribute.map` takes a list of LDAP attributes and their mapped values in the format `attribute_name|value`.

`attr_session_map`
: Takes a value based on the `am_map_t` (defined in the `<am_map.h>` header file) that represents one or more session attributes and a corresponding value. This member is enabled when the following two properties in `AMAgent.properties` are configured:

- `com.sun.am.policy.agents.config.session.attribute.fetch.mode` takes a value of `HTTP_HEADER` or `HTTP_COOKIE`.
- `com.sun.am.policy.agents.config.session.attribute.map` takes a list of session attributes and their mapped values in the format `attribute_name|value`.

`attr_response_map`
: Takes a value based on the `am_map_t` (defined in the `<am_map.h>` header file) that represents one or more response attributes and a corresponding value. This member is enabled when the following two properties in `AMAgent.properties` are configured:

- `com.sun.am.policy.agents.config.response.attribute.fetch.mode` takes a value of `HTTP_HEADER` or `HTTP_COOKIE`.
- `com.sun.am.policy.agents.config.response.attribute.map` takes a list of response names and their mapped values in the format `attribute_name|value`.

`advice_string`
: Pointer to a string that defines a value for further authentication if dictated by the policy condition.If no condition is specified, the advice string will have an empty value.

Memory Concerns

Memory for `am_policy_result_t` is allocated by `am_policy_evaluate()` and freed by `am_policy_result_destroy()`.

am_policy_t

Declares an unsigned integer as a type for a policy object.

Syntax

```
#include "am_policy.h"
typedef unsigned int am_policy_t;
```

Members

am_policy_t has no members.

am_resource_traits_t

Contains the functions to return resource traits that will be used to compare with a user's defined policy and evaluate the access request.

Syntax

```
#include "am_policy.h"
typedef struct am_resource_traits {
    am_resource_match_t (*cmp_func_ptr)(const struct am_resource_traits *rsrc_traits,
                                        const char *policy_res_name,
                                        const char *resource_name,
                                        boolean_t use_patterns);
    boolean_t (*has_patterns)(const char *resource_name);
    boolean_t (*get_resource_root)(const char *resource_name,
                                   char *root_resource_name,
                                   size_t buflength);
    boolean_t ignore_case;
    char separator;
    void (*canonicalize)(const char *resource, char **c_resource);
    void (*str_free)(void *resource_str);
} am_resource_traits_t;
```

Members

cmp_func_ptr

Pointer to a function that compares policy_res_name and resource_name to return one of the following values of the am_resource_match_t enumeration (defined in the <am_policy.h> header file):

```
typedef enum am_resource_match {
    AM_SUB_RESOURCE_MATCH,
```

```
    AM_EXACT_MATCH,
    AM_SUPER_RESOURCE_MATCH,
    AM_NO_MATCH,
    AM_EXACT_PATTERN_MATCH
} am_resource_match_t;
```

Tip – `cmp_func_ptr` can point to `am_policy_compare_urls()` to evaluate URL resources.

`rsrc_traits`	Pointer to the resource traits structure containing data regarding a policy.
`policy_res_name`	Pointer to the name of the resource being protected.
`resource_name`	Pointer to the name of the resource being requested.
`use_patterns`	Based on the `boolean_t` defined in the `<am_types.h>` header file, `B_TRUE` indicates that the function will use or recognize patterns when comparing resources.

`has_patterns`

Pointer to a function that determines whether a resource has patterns and returns one of the following values of the `boolean_t` enumeration defined in the `<am_types.h>` header file:

`B_TRUE`	If `resource_name` has patterns.
`B_FALSE`	Otherwise.

Tip – `has_patterns` can point to `am_policy_resource_has_patterns()` for URL resources.

`resource_name`	Pointer to the name of the resource being requested.

`get_resource_root`

Pointer to a function that extracts the root of the specified resource and returns one of the following values of the `boolean_t` enumeration defined in the `<am_types.h>` header file:

`B_TRUE`	If the resource root was successfully inserted into the specified `root_resource_name` buffer.
`B_FALSE`	Otherwise.

Tip – `get_resource_root` can point to `am_policy_get_url_resource_root()` for URL resources.

`resource_name`	Pointer to the name of the resource being requested.
`root_resource_name`	Buffer to hold the resource root.
`buflength`	Value based on the `size_t` defined in the standard `<stddef.h>` header file that reflects the length of the `root_resource_name` buffer.

`ignore_case`

Value that takes one of the following values of the `boolean_t` enumeration defined in the `<am_types.h>` header file:

B_TRUE	Ignore case for all functions in this structure.
B_FALSE	Otherwise.

separator
Defines the resource separator. For URLs / should be used.

canonicalize
Pointer to a function that converts the specified resource name into a standard representation for comparative purposes.

resource	Pointer to a resource name. This could be the resource being requested or the resource defined in the policy.
c_resource	Output of the canonical resource name.

Note – Memory for the canonical name must be allocated by the caller. A function to free the allocated memory must be set in str_free.

str_free
Pointer to a function to free a c_resource string after the results have been evaluated by am_policy_evaluate(). This field cannot be set to NULL.

Note – free() should be used if canonicalize is set to the am_policy_resource_canonicalize() function.

resource_str	Pointer to the string returned in the canonicalize function.

Policy Functions

The functions defined in <am_policy.h> are:

am_policy_compare_urls()

Compares the URLs of two resources, and returns the appropriate result.

Syntax

```
#include "am_policy.h"
AM_EXPORT am_resource_match_t
am_policy_compare_urls(const am_resource_traits_t *rsrc_traits,
                       const char *policy_resource_name,
                       const char *resource_name,
                       boolean_t use_patterns);
```

Parameters

This function takes the following parameter:

`rsrc_traits` — Pointer to a `am_resource_traits_t` type containing data regarding a policy.

Note – See "am_resource_traits_t" on page 49 for more information.

`policy_resource_name` — Pointer to the name of the resource being protected.

`resource_name` — Pointer to the name of the resource being requested.

`use_patterns` — Based on the `boolean_t` defined in the <am_types.h> header file, `B_TRUE` indicates that the function will consider occurrences of * in `policy_resource_name` as wild cards. If `B_FALSE`, occurrences of * are taken as a literal characters.

Note – In cases of `SUB_RESOURCE_MATCH` and `SUPER_RESOURCE_MATCH` when `usePatterns` is `B_TRUE`, the patterns are sub or super matching patterns, respectively.

Returns

This function returns one of the following values of the `am_resource_match_t` enumeration as defined:

```
#include "am_policy.h"
typedef enum am_resource_match {
    AM_SUB_RESOURCE_MATCH,
    AM_EXACT_MATCH,
    AM_SUPER_RESOURCE_MATCH,
    AM_NO_MATCH,
```

```
    AM_EXACT_PATTERN_MATCH
} am_resource_match_t;
```

`AM_EXACT_MATCH`	If both URLs match exactly as in, for example, if the URL for the resource is `http://example.sun.com:90/index.html` and the URL in the policy is `http://example.sun.com:90/index.html`.
`AM_EXACT_PATTERN_MATCH`	This result is returned if the URL to which the policy applies matches the URL to which access is requested as in, for example, if the URL for the resource is `http://example.sun.com:90/index.html` and the URL in the policy is `http://example.sun.com:90/*`. Distinction is not made between an `EXACT_MATCH` or a pattern match.
`AM_NO_MATCH`	If the URLs do not match.
`AM_SUB_RESOURCE_MATCH`	If the requested URL is found to be a sub-resource of the URL defined in the policy.
`AM_SUPER_RESOURCE_MATCH`	If the requested URL is found to be a parent of the URL defined in the policy.

am_policy_destroy()

Destroys an initialized instance of a policy evaluator object.

Details

An instance is initialized for each policy request.

Note – The caller must ensure that the same instance is not destroyed more than once.

Syntax

```
#include "am_policy.h"
AM_EXPORT am_status_t
am_policy_destroy(am_policy_t policy);
```

Parameters

This function takes the following parameter:

`policy`	Integer specifying the object being destroyed.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the call was successful.
AM_*	If any error occurs, the type of error indicated by the status value.

am_policy_evaluate()

Evaluates a policy for a given request and returns a non-boolean result.

Note – am_policy_evaluate() has been deprecated. See "am_policy_evaluate_ignore_url_notenforced()" on page 56.

Details

am_policy_evaluate() was used to evaluate policy for URLs on the not-enforced list and those not on the not-enforced list. Since there is not a need to evaluate URLs on the not-enforced list, am_policy_evaluate() has been deprecated. Although it can still be used, the SDK invokes am_policy_evaluate_ignore_url_notenforced().

Syntax

```
#include "am_policy.h"
AM_EXPORT am_status_t
am_policy_evaluate(am_policy_t policy_handle,
                   const char *sso_token,
                   const char *resource_name,
                   const char *action_name,
                   const am_map_t env_parameter_map,
                   am_map_t policy_response_map_ptr,
                   am_policy_result_t *policy_result);
```

Parameters

This function takes the following parameters:

policy_handle	Integer specifying the object being evaluated.
sso_token	Pointer to the session token (SSOTokenID) of the authenticated user.

Note – The Access Manager Session Service creates a session data structure (also known as an SSOToken) that stores information such as login time, authentication scheme, and authentication level. It also generates a session token (also known as an SSOTokenID, a randomly-generated string that identifies an instance of an SSOToken.

resource_name	Pointer to the name of the resource being requested.
action_name	Pointer to the action requested.

Note – An *action* is the operation to be performed on the resource. Web server actions are POST and GET. An allowable action for a human resources service , for example, can change a home telephone number.

env_parameter_map	Map object which contains environment variables (IP address, host name, etc.) used for evaluation by the Policy Service.

Note – See "am_map_t" on page 113 for more information.

policy_response_map_ptr	Pointer to a map object which contains all the profile, session and response attributes fetched.

Note – This must be enabled in AMAgent.properties. See "am_policy_result_t" on page 47 for information on how this is done. See "am_map_t" on page 113 for more information on map objects.

policy_result	Pointer to the am_policy_result_t type to store the result.

Note – See "am_policy_result_t" on page 47 for more information.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the call was successful.
AM_*	If any error occurs, the type of error indicated by the status value.

Memory Concerns

After using the results the caller must call `am_policy_result_destroy()` on `policy_result` to cleanup the allocated memory. Also, `am_map_destroy()` must be called on `policy_response_map_ptr` and `env_parameter_map` after their respective usage.

am_policy_evaluate_ignore_url_notenforced()

Evaluates a policy for a given request and returns a non-boolean result.

Details

`am_policy_evaluate_ignore_url_notenforced()` will evaluate a policy for the specified URL only if the URL does not appear on the not-enforced list defined in `AMAgent.properties`.

Note – See *Sun Java System Access Manager Policy Agent 2.2 User's Guide* for more information.

Syntax

```
#include "am_policy.h"
AM_EXPORT am_status_t
am_policy_evaluate_ignore_url_notenforced(am_policy_t policy_handle,
                                          const char *sso_token,
                                          const char *resource_name,
                                          const char *action_name,
                                          const am_map_t env_parameter_map,
                                          am_map_t policy_response_map_ptr,
                                          am_policy_result_t *policy_result,
                                          am_bool_t ignorePolicyResult),
                                          char **am_revision_number;
```

Parameters

This function takes the following parameters:

`policy_handle`	Integer specifying the object being evaluated.
`sso_token`	Pointer to the session token (`SSOTokenID`) of the authenticated user.

Note – The Access Manager Session Service creates a session data structure (also known as an `SSOToken`) that stores information such as login time, authentication scheme, and authentication level. It also generates a session token (also known as an `SSOTokenID`, a randomly-generated string that identifies an instance of an `SSOToken`.

`resource_name`	Pointer to the name of the resource being requested.
`action_name`	Pointer to the action requested.

Note – An *action* is the operation to be performed on the resource. Web server actions are `POST` and `GET`. An allowable action for a human resources service , for example, can change a home telephone number.

`env_parameter_map`	Map object which contains environment variables (IP address, host name, etc.) used for evaluation by the Policy Service.

Note – See “`am_map_t`” on page 113 for more information.

`policy_response_map_ptr`	Pointer to a map object which contains all the profile, session and response attributes fetched.

Note – This must be enabled in `AMAgent.properties`. See “`am_policy_result_t`” on page 47 for information on how this is done. See “`am_map_t`” on page 113 for more information on map objects.

`policy_result`	Pointer to the `am_policy_result_t` type to store the result.

Note – See “`am_policy_result_t`” on page 47 for more information.

`ignorePolicyResult`	Based on the `am_bool_t` defined in the `<am_types.h>` header file, `AM_TRUE` indicates that policy evaluation will not be done for the URL.
`am_revision_number`	Takes a value equal to the version of the instance of Access Manager with which the SDK is communicating. When communicating with Access Manager 7.0, the value will be 7.0, otherwise 6.3. It can also be set to `NULL`.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS If the call was successful.

AM_* If any error occurs, the type of error indicated by the status value.

Memory Concerns

After using the results the caller must call am_policy_result_destroy() on policy_result to cleanup the allocated memory. Also, am_map_destroy() must be called on policy_response_map_ptr and env_parameter_map after their respective usage.

am_policy_get_url_resource_root()

Extracts the root of a given URL.

Details

am_policy_get_url_resource_root() populates the resource_root pointer with the extracted information. For example, http://www.sun.com/index.html will return http://www.sun.com/ and http://www.sun.com:8080/index.html will return http://www.sun.com:8080/.

Syntax

```
#include "am_policy.h"
AM_EXPORT boolean_t
am_policy_get_url_resource_root(const char *resource_name,
                                      char *resource_root,
                                size_t length);
```

Parameters

This function takes the following parameters:

resource_name Pointer to the protected resource URL.

resource_root Pointer to the location where the resource root will be written.

length Value based on the size_t defined in the standard <stddef.h> header file that reflects the size of the resource_root buffer.

Note – When using resources other than URLs, the developer implementing this function must make accurate judgement about the minimum size of `resource_root`.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the <am_types.h> header file):

`B_TRUE`	If the root was successfully extracted.
`B_FALSE`	If not.

am_policy_init()

Initializes the Access Manager Policy Service.

Syntax

```
#include "am_policy.h"
AM_EXPORT am_status_t
am_policy_init(am_properties_t policy_config_properties);
```

Parameters

This function takes the following parameter:

`policy_config_properties`	Pointer to the properties used to initialize the Policy Service.

Note – See “`am_properties_t`” on page 131 for more information.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the <am_types.h> header file):

`AM_SUCCESS`	If the call was successful.
`AM_*`	If any error occurs, the type of error indicated by the status value.

Memory Concerns

The caller must call `am_policy_destroy()` to free the memory.

am_policy_invalidate_session()

Cancels the specified session.

Syntax

```
#include "am_policy.h"
AM_EXPORT am_status_t
am_policy_invalidate_session(am_policy_t policy_handle,
                             const char *ssoTokenId);
```

Parameters

This function takes the following parameters:

`policy_handle`	Integer specifying the object being evaluated.
`ssoTokenId`	Pointer to the session token of the authentication user.

Note – The *session token* is a randomly-generated string that represents an authenticated user.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the <am_types.h> header file):

`AM_SUCCESS`	If the call was successful.
`AM_*`	If any error occurs, the type of error indicated by the status value.

am_policy_is_notification_enabled()

Checks whether the notification functionality is enabled.

Syntax

```
#include "am_policy.h"
AM_EXPORT boolean_t
am_policy_is_notification_enabled(am_policy_t policy_handle);
```

Parameters

This function takes the following parameter:

`policy_handle`	Integer specifying the object being evaluated.

Returns

This function returns the standard boolean_t with one of the following values:

0	If notification is disabled.
non-zero	If notification is enabled.

am_policy_notify()

Refreshes the policy cache when a policy notification is received by the client.

Syntax

```
#include "am_policy.h"
AM_EXPORT am_status_t
am_policy_notify(am_policy_t policy_handle,
                 const char *notification_data,
                 size_t notification_data_len);
```

Parameters

This function takes the following parameters:

policy_handle	Integer specifying the object being evaluated.
notification_data	Pointer to the notification message as an XML string.
notification_data_len	Value based on the size_t defined in the standard <stddef.h> header file that reflects the size of the notification_data string.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the call was successful.
AM_*	If any error occurs, the type of error indicated by the status value.

am_policy_resource_canonicalize()

Converts the specified resource name into a standard representation for comparison purposes.

Syntax

```
#include "am_policy.h"
AM_EXPORT void
am_policy_resource_canonicalize(const char *resource,
                                      char **c_resource);
```

Parameters

This function takes the following parameters:

`resource`	Pointer to the name of the resource to be converted.
`c_resource`	Pointer to a pointer to the location where the converted string will be placed.

Returns

This function does not return a value.

am_policy_resource_has_patterns()

Checks whether the specified resource name has patterns (such as the wildcard *).

Syntax

```
#include "am_policy.h"
AM_EXPORT boolean_t
am_policy_resource_has_patterns(const char *resource_name);
```

Parameters

This function takes the following parameter:

`resource_name`	Pointer to the resource being evaluated.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the <am_types.h> header file):

`B-TRUE`	If the resource has patterns.
`B_FALSE`	Otherwise.

am_policy_result_destroy()

Destroys the specified am_policy_result_t structure type.

Note – See "am_policy_result_t" on page 47 for more information.

Syntax

```
#include "am_policy.h"
AM_EXPORT void
am_policy_result_destroy(am_policy_result_t *result);
```

Parameters

This function takes the following parameter:

result	Pointer to the am_policy_result_t structure type being destroyed.

Returns

This function does not return a value.

am_policy_service_init()

Initializes one instance of Access Manager Policy Service for policy evaluation.

Syntax

```
#include "am_policy.h"
AM_EXPORT am_status_t
am_policy_service_init(const char *service_name,
                       const char *instance_name,
                       am_resource_traits_t rsrc_traits,
                       am_properties_t service_config_properties,
                       am_policy_t *policy_handle_ptr);
```

Parameters

This function takes the following parameters:

service_name	Pointer to the name for the Policy Service.
instance_name	Pointer to the name of the instance being initialized.
rsrc_traits	Pointer to a am_resource_traits_t structure type.

	Note – See "am_resource_traits_t" on page 49 for more information.
service_config_properties	Pointer to the properties used to initialize the Policy Service instance. **Note** – See "am_properties_t" on page 131 for more information.
policy_handle_ptr	Pointer to the integer specifying the object being evaluated.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the call was successful.
AM_*	If any error occurs, the type of error indicated by the status value.

CHAPTER 4

Single Sign-On Data Types and Functions

Sun Java™ System Access Manager contains public data types and functions you can use to communicate with the Session Service for single sign-on. Reference summaries include a short description, syntax, parameters and returns. The code is contained in the <am_sso.h> header file (located in the `/`*AccessManager-base*`/SUNWam/include` directory). The sample source `am_sso_test.c` (located in the `/`*AccessManager-base*`/SUNWam/samples/csdk` directory) demonstrates the basic usage of the single sign-on API. This chapter contains the following sections:

- "The Single Sign-on API for C" on page 65
- "Single Sign-on Data Types" on page 73
- "Single Sign-on Functions" on page 74

The Single Sign-on API for C

The Single Sign-on API for C are provided in the `SUNWamcom` package which comes with Access Manager or any of its downloadable policy agents. The package includes header files, libraries and samples. The header files are:

- <am_sso.h> which must be included for any single sign-on routines.
- <am_notify.h> which must be included for parsing notification messages from the server and calling single sign-on listeners.

Single Sign-on Properties

Certain properties must be read and passed to `am_sso_init()` in order to initialize the Session Service. Thus, `am_sso_init()` must be called before any other single sign-on interface. By default, the properties file used for initializing the Session Service is `AMAgent.properties`, located in `/`*AccessManager-base*`/SUNWam/config/`. This file is created during the process for installing a web agent. Before using the API be sure the properties in the following table are set in `AMAgent.properties`.

Note – See *Sun Java System Access Manager Policy Agent 2.2 User's Guide* for more information.

TABLE 4–1 Single Sign-on Properties in `AMAgent.properties`

Property	Definition
`com.sun.am.naming.url`	Specifies the URL for the Naming Service which, in turn, finds the URL of the Session Service. This property must be set as: **`com.sun.am.naming.url =`** *protocol*`://`*AM_host*`.`*domain*`:`*port*`/`**`amserver/namingservice`**
`com.sun.am.notification.enable`	Specifies whether the Notification Service will be used to update the cache. If enabled, a URL where notification messages from Access Managerare sent must be specified. This property is set as: **`com.sun.am.notification.enable =`** *true* | *false* **Note** – If `com.sun.am.notification.enable` is not found in the properties file, the default value is `false`.
`com.sun.am.notification.url`	If `com.sun.am.notification.enable` is set to true, the value of this property specifies a URL where notification messages from Access Managerare sent. This property is set as: **`com.sun.am.notification.url =`** *protocol*`//`*AM_host*`.`*domain*`:`*port*`/`*notification_URL*
`com.sun.am.sso.polling.period`	Specifies how often, in minutes, the cache should be checked for entries that have reached the cache entry life time. This property must be set as: **`com.sun.am.sso.checkCacheInterval=`***#* **Note** – By default, this property is not in `AMAgent.properties` but can be added when needed.
`com.sun.am.sso.max_threads`	Specifies the maximum number of threads the single sign-on API for C should invoke for handling notifications. The API maintains a thread pool and invokes a thread for each notification. If the maximum number of threads has been reached, the notification will wait until a thread is available. This property must be set as: **`com.sun.am.sso.maxThreads=`***#* If not specified the default maximum number of threads is 10. **Note** – By default, this property is not in `AMAgent.properties` but can be added when needed.

For more information, see the *Sun Java System Access Manager Policy Agent 2.2 User's Guide*.

Single Sign-on Calls

The following sections contain information and code samples for some of the single sign-on calls.

Initialization and Cleanup

When implementing single sign-on, `am_sso_init()` must be called before any other `am_sso_*` functions to initialize the internal data structures. At the end of all single sign-on routines, `am_cleanup()` should be called for cleanup. Following is a code sample using these functions.

Note – For more information on `am_cleanup()`, see Chapter 9.

```
#include <am_sso.h>

     int main() {
        am_properties_t *properties;
        am_status_t status;

        /* create a properties handle */
        status = am_properties_create(&properties);
        if (status != AM_SUCCESS) {
             printf("am_properties_create failed.\\n");
             exit(1);
        }

        /* load properties from a properties file */
        status = am_properties_load(properties, "./myPropertiesFile");
        if (status != AM_SUCCESS) {
            printf("am_properties_load failed.\\n");
            exit(1);
        }

        /* initialize SSO module */
        status = am_sso_init(properties);
        if (status != AM_SUCCESS) {
            printf("am_sso_init failed.\\n");
            return 1;
        }

         /* login through auth module, and do auth functions.
          * ...
          */
```

```
    /* do sso functions
     * ...
     */

     /* done - cleanup. */
     status = am_cleanup();
     if (status != AM_SUCCESS) {
         printf("am_cleanup failed!\\n");
         return 1;
     }
     /* free memory for properties */
     status = am_properties_destroy(properties);
     if (status != AM_SUCCESS) {
         printf("Failed to free properties.\\n");
         return 1;
     }

     /* exit program successfully. */
     return 0;
}
```

Single Sign-on Token Handles

When a user attempts to access a protected resource, the Session Service creates a new, empty session data structure (also known as an `SSOToken`) that will store information (such as login time, authentication scheme, authentication level, maximum time out limits and caching time limits) after the user is successfully authenticated. Additionally, the Session Service generates a session identifier (also known as an `SSOTokenID`) which is a randomly-generated string that identifies the user and corresponding session structure. Technically, the `SSOTokenID` identifies an instance of an `SSOToken`.

After a user has been successfully authenticated, the `SSOToken` is activated and the relevant session information is stored in the structure. Additionally, the state of the `SSOTokenID` is changed from invalid to valid. When using the single sign-on API for C, a *single sign-on token handle* contains this valid `SSOTokenID` and allows for operations based on the `SSOToken`.

- "Creating Single Sign-on Token Handles" on page 68
- "Validating Single Sign-on Token Handles" on page 69
- "Destroying Session Token Handles" on page 69

Creating Single Sign-on Token Handles

Once activated, an `SSOToken` can be obtained and inserted into a single sign-on token handle by passing the `SSOTokenID` to `am_sso_create_sso_token_handle()`. This function then checks to see if the identifier is in its local cache and, if not, retrieves the session information associated with the `SSOTokenID` from Access Manager and caches it. A single sign-on token handle is then assigned to it.

Validating Single Sign-on Token Handles

The caller can check if the session is valid using `am_sso_is_valid_token()`. If not valid, `am_sso_validate_token()` will flush the old session information from the local cache (if any) and fetch the latest session information from Access Manager.

Note – `am_sso_refresh_token()` duplicates the functionality of `am_sso_validate_token()`. In addition, it will reset the idle time of the session on the server.

Destroying Session Token Handles

When the caller is finished with a token handle, it must be freed to prevent memory leak by calling `am_sso_destroy_sso_token_handle()`. The session associated with the token handle can be invalidated or ended with `am_sso_invalidate_token()`.

Tip – Although this ends the session for the user, the proper way to log out is by using `am_auth_logout()` as described in "`am_auth_logout()`" on page 43. Not using `am_auth_logout()` will result in authentication resources associated with the session remaining on the server unnecessarily until the session has timed out.

Retrieving and Setting Properties

The following code sample shows how you might use the `am_sso_get_property()` and `am_sso_set_property()` functions. For additional information, see "`am_sso_get_property()`" on page 83 and "`am_sso_set_property()`" on page 89.

```
/* initialize sso as in previous sample */

     am_status_t status = NULL;
     am_sso_token_handle_t sso_handle = NULL;
     char *session_status = NULL;
     am_string_set_t principal_set = NULL;

     /* create sso token handle */
     status = am_sso_create_sso_token_handle(&sso_handle, sso_token_id, false);
     if (status != AM_SUCCESS) {
         printf("Failed getting sso token handle for sso token id %s.
                    \\n", sso_token_id);
         return 1;
     }

     /* check if session is valid */
     session_status = am_sso_is_valid_token(sso_handle) ? "Valid" : "Invalid";
     printf("Session state is %s\\n", session_status);

     /* check if session is valid using validate. This also updates the handle with
```

```
    /*info from the server */
status = am_sso_validate_token(sso_handle);
if (status == AM_SUCCESS) {
    printf("Session state is valid.\\n");
} else if (status == AM_INVALID_SESSION) {
    printf("Session status is invalid.\\n");
} else {
    printf("Error validating sso token.\\n");
    return 1;
}

/* get info on the session */
printf("SSO Token ID is %s.\\n", am_sso_get_sso_token_id(sso_handle));
printf("Auth type is %s.\\n", am_sso_get_auth_type(sso_handle));
printf("Auth level is %d.\\n", am_sso_get_auth_level(sso_handle));
printf("Idle time is %d.\\n", am_sso_get_idle_time(sso_handle));
printf("Max Idle time is %d.\\n", am_sso_get_max_idle_time(sso_handle));
printf("Time left is %d.\\n", am_sso_get_time_left(sso_handle));
printf("Max session time is %d.\\n", am_sso_get_max_session_time(sso_handle));
printf("Principal is %s.\\n", am_sso_get_principal(sso_handle));
printf("Host is %s.\\n", am_sso_get_host(sso_handle));
principal_set = am_sso_get_principal_set(sso_handle);
if (principal_set == NULL) {
       printf("ERROR: Principal set is NULL!\\n");
}else {
       printf("Principal set size %d.\\n", principal_set->size);
       for (i = 0; i < principal_set->size; i++) {
           printf("Principal[%d] = %s.\\n", i, principal_set->strings[i]);
       }
       am_string_set_destroy(principal_set);
}

/* get "HOST" property on the session. Same as am_sso_get_host(). */
printf("Host is %s.\\n", am_sso_get_property(sso_handle, "HOST"));

/* set a application defined property and get it back */
status = am_sso_set_property(sso_handle, "AppPropName", "AppPropValue");
if (status != AM_SUCCESS) {
    printf("Error setting property.\\n");
    return 1;
}
printf("AppPropName value is %s.\\n", am_sso_get_property
           (sso_handle, "AppPropName");

/* refresh token, idle time should be 0 after refresh */
status = am_sso_refresh_token(sso_handle);
if (status != AM_SUCCESS) {
    printf("Error refreshing token !\\n");
```

```
        return 1;
    }
    printf("After refresh, idle time is %d.\\n", am_sso_get_idle_time(sso_handle));

    /* end this session abruptly. am_auth_logout() is the right way
        /* to end session */
    status = am_sso_invalidate_token(sso_handle);
    if (status != AM_SUCCESS) {
        printf("Error invalidating token.\\n");
        return 1;
    }

    /* we are done with sso token handle. free memory for sso handle. */
    status = am_sso_destroy_sso_token_handle(sso_handle);
    if (status != AM_SUCCESS) {
        printf("Failed to free sso token handle.\\n");
        return 1;
    }

    /* call am_cleanup, and other cleanup routines as in previous sample */
```

Listening and Notification

A session may become invalid because it has been idle over a time limit, it has reached the maximum session time, or it has been terminated by an administrator. An application can be notified of this by implementing a listener function. Additionally, notification must be enabled for the application to receive change notifications when `am_sso_init()` is initialized. Notification is enabled by setting the `com.sun.am.notification.enable` property in `AMAgent.properties` to `true`, and by providing the `com.sun.am.notification.url` property a URL which will receive HTTP notification messages from Access Manager. Notification messages are in XML and should be passed as a string (`const char *`) to `am_notify()` which will parse the message and invoke the appropriate session or policy listener. Following is a code sample that illustrates this.

Note – For more information, see "Single Sign-on Properties" on page 65 and "<am_notify.h>" on page 206.

```
void sample_listener_func(
                   am_sso_token_handle_t sso_token_handle,
                   const am_sso_token_event_type_t event_type,
                   const time_t event_time,
                   void *opaque)
     {
         if (sso_token_handle != NULL) {
             const char *sso_token_id = am_sso_get_sso_token_id(sso_token_handle);
             boolean_t is_valid = am_sso_is_valid_token(sso_token_handle);
             printf("sso token id is %s.\\n",
                    sso_token_id==NULL?"NULL":sso_token_id);
```

```
            printf("session state is %s.\\n",
                     is_valid == B_TRUE ? "valid":"invalid");
            printf("event type %d.\\n", event_type);
            printf("event time %d.\\n", event_time);
        }
        else {
            printf("Error: sso token handle is null!");
        }
        if (opaque)
            *(int *)opaque = 1;
        return;
    }

    int main(int argc, char *argv[]) {

    am_status_t status;
    char *sso_token_id = argv[1];
    int listener_func_done = 0;

    /* initialize sso as in previous samples */

    /* get sso token handle */
    status = am_sso_create_sso_token_handle(&sso_handle, sso_token_id,  false);

    /* register listener function. notification must be enabled, if not,
    /* status AM_NOTIF_NOT_ENABLED will be returned. */
status = am_sso_add_sso_token_listener(sso_handle, sample_listener_func,
               &listener_func_done, B_TRUE);
    if (status != AM_SUCCESS) {
        printf("Failed to register sample listener function.\\n");
        return 1;
    }
```

Non-Web Applications

Access Manager provides the single sign-on API for C to be used primarily with web-based applications. It can though be extended to non-web applications with limitations. You can use the API with non-web applications in either of the following ways:

- The application has to obtain the Access Manager cookie value and pass it to the single sign-on client methods to retrieve the `SSOToken`. The method used for this process is application-specific.
- Command line applications, such as `amadmin`, can be used. Session tokens can be created to access Directory Server directly. No session is created, making Access Manager access valid only within that process or virtual machine.

Single Sign-on Data Types

The single sign-on data types defined in <am_sso.h> are:

- "am_sso_token_handle_t" on page 73
- "am_sso_token_listener_func_t" on page 73

am_sso_token_handle_t

A pointer to the session information object.

Syntax

```
#include "am_sso.h"
typedef struct am_sso_token_handle *am_sso_token_handle_t;
```

Members

am_sso_token_handle is an opaque structure with no accessible members.

am_sso_token_listener_func_t

Listener function declaration.

Syntax

```
#include "am_sso.h"
typedef void (*am_sso_token_listener_func_t)(
                 const am_sso_token_handle_t sso_token_handle,
                 const am_sso_token_event_type_t event_type,
                 const time_t event_time,
                 void *args);
```

Members

am_sso_token_listener_func_t has the following members:

sso_token_handle — Pointer to the session information object.

event_type — Takes one of the following values from the am_sso_token_event_type_t enumeration (defined in the <am_sso.h> header file):

```
typedef enum {
    AM_SSO_TOKEN_EVENT_TYPE_UNKNOWN = 0,
    AM_SSO_TOKEN_EVENT_TYPE_IDLE_TIMEOUT = 1,
    AM_SSO_TOKEN_EVENT_TYPE_MAX_TIMEOUT = 2,
```

```
    AM_SSO_TOKEN_EVENT_TYPE_LOGOUT = 3,
    AM_SSO_TOKEN_EVENT_TYPE_DESTROY = 5
} am_sso_token_event_type_t;
```

event_time	Takes a value based on the standard time_t data type that represents the time at which the change event occurred.
*args	Pointer to application-defined parameters.

Single Sign-on Functions

The single sign-on functions defined in <am_sso.h> are:

am_sso_add_listener()

Add a listener for any and all event changes related to the referenced single sign-on token handle.

Note – am_sso_add_listener() will not be removed after it is called once like "am_sso_add_sso_token_listener()" on page 76.

Details

The caller must do one of the following:

- Provide a URL to this function.
- Enable notification and provider a valid notification URL in the `AMAgent.properties` file passed to `am_sso_init()`.

See "Listening and Notification" on page 71 for more information.

Syntax

```
#include "am_sso.h"
AM_EXPORT am_status_t
am_sso_add_listener(const am_sso_token_listener_func_t listener,
                    void *args,
                    boolean_t dispatch_to_sep_thread);
```

Parameters

This function takes the following parameters:

`listener`	The listener as described in "`am_sso_token_listener_func_t`" on page 73. **Note –** When the listener is called, updated session information from Access Manager is passed in a temporary `sso_token_handle`.
`args`	Pointer to application-defined arguments to pass to the listener.
`dispatch_to_sep_thread`	Takes one of the values based on the `boolean_t` (defined in the `<am_types.h>` header file) that indicates whether the listener function should be called in the calling thread or dispatched to a thread from the internal thread pool managed by the C SDK. **Note –** Calling the listener in a thread from an internal thread pool allows `am_notify()` to return immediately upon parsing the notification message rather than waiting for the listener functions to finish before returning.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the listener was successfully added.

`AM_INVALID_ARGUMENT`	If `sso_token_handle` or listener is invalid, or the notification URL is not set and none is provided in the properties file.
`AM_NOTIF_NOT_ENABLED`	If notification is not enabled and the notification URL parameter is invalid.
`AM_FAILURE`	If any other error occurred.

`am_sso_add_sso_token_listener()`

Adds a listener for any and all event changes related to the referenced single sign-on token handle.

Note – `am_sso_add_sso_token_listener()` is removed from memory after it is called once, differentiating its functionality from “`am_sso_add_listener()`” on page 74.

Details

The caller must do one of the following:

- Provide a URL to this function.
- Enable notification and provider a valid notification URL in the `AMAgent.properties` file passed to `am_sso_init()`.

See “Listening and Notification” on page 71 for more information.

Syntax

```
#include "am_sso.h"
AM_EXPORT am_status_t
am_sso_add_sso_token_listener(am_sso_token_handle_t sso_token_handle,
                              const am_sso_token_listener_func_t listener,
                              void  *args,
                              boolean_t dispatch_to_sep_thread);
```

Parameters

This function takes the following parameters:

`sso_token_handle`	Pointer to the session information object containing the session token to which the listener corresponds. The handle will be filled with the session information from the notification message, overwriting any existing contents.

	Note – The session token is a randomly-generated string that represents an authenticated user.
`listener`	The listener as described in “`am_sso_token_listener_func_t`” on page 73.
	Note – When the listener is called, updated session information from Access Manager is passed in a temporary `sso_token_handle`.
`args`	Arguments to pass to the listener.
`dispatch_to_sep_thread`	Takes one of the values based on the `boolean_t` (defined in the `<am_types.h>` header file) that indicates whether the listener function should be called in the calling thread or dispatched to a thread from the internal thread pool managed by the C SDK.
	Note – Calling the listener in a thread from an internal thread pool allows `am_notify()` to return immediately upon parsing the notification message rather than waiting for the listener functions to finish before returning.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the listener was successfully added.
`AM_INVALID_ARGUMENT`	If `sso_token_handle` or listener is invalid, or the notification URL is not set and none is provided in the properties file.
`AM_NOTIF_NOT_ENABLED`	If notification is not enabled and the notification URL parameter is invalid.
`AM_FAILURE`	If any other error occurred.

`am_sso_create_sso_token_handle()`

Creates a single sign-on token handle as a container for a valid `SSOTokenID`.

Details

For more information, see “Single Sign-on Token Handles” on page 68.

Syntax

```
#include "am_sso.h"
AM_EXPORT am_status_t
am_sso_create_sso_token_handle(am_sso_token_handle_t *sso_token_handle_ptr,
                               const char *sso_token_id,
                               boolean_t reset_idle_timer);
```

Parameters

This function takes the following parameters:

`sso_token_handle`	Pointer to a `am_sso_token_handle_t` type which will be assigned if the session validation is successful.
`sso_token_id`	Pointer to the `SSOTokenID` to which the handle will be associated.
`reset_idle_timer`	Takes one of the values based on the `boolean_t` (defined in the `<am_types.h>` header file) that specifies that the idle time of the `SSOTokenID` on the server will be refreshed when querying for session information.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If session validation was successful and a single sign-on token handle was successfully created.
`AM_SERVICE_NOT_INITIALIZED`	If the Session Service was not initialized.
`AM_INVALID_ARGUMENT`	If the `session_token_handle_ptr` parameter is `NULL`.
`AM_NO_MEMORY`	If there was a memory allocation problem.
`AM_FAILURE`	If any other error occurred.

`am_sso_destroy_sso_token_handle()`

Destroys the specified single sign-on token handle.

Details

`am_sso_destroy_sso_token_handle()` does not log out the user or invalidate the session. For more information, see “Single Sign-on Token Handles” on page 68.

Syntax

```
#include "am_sso.h"
AM_EXPORT am_status_t
am_sso_destroy_sso_token_handle(am_sso_token_handle_t sso_token_handle);
```

Parameters

This function takes the following parameter:

`sso_token_handle`	Pointer to a `am_sso_token_handle_t` type which will be destroyed.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the memory release process was successful.
`AM_INVALID_ARGUMENT`	If the `sso_token_handle` parameter is `NULL`.
`AM_FAILURE`	If any other error occurred.

am_sso_get_auth_level()

Retrieves the authentication level associated with the specified single sign-on token handle.

Syntax

```
#include "am_sso.h"
AM_EXPORT unsigned long
am_sso_get_auth_level(const am_sso_token_handle_t sso_token);
```

Parameters

This function takes the following parameter:

`sso_token`	Pointer to a `am_sso_token_handle_t` type.

Returns

This function returns the authentication level of the specified handle, or `ULONG_MAX` if an error occurred.

am_sso_get_auth_type()

Retrieves the authentication type associated with the specified single sign-on token handle.

Syntax

```
#include "am_sso.h"
AM_EXPORT const char *
am_sso_get_auth_type(const am_sso_token_handle_t sso_token);
```

Parameters

This function takes the following parameter:

`sso_token` Pointer to a `am_sso_token_handle_t` type.

Returns

This function returns the authentication type of the specified handle, or `NULL` if an error occurred.

am_sso_get_host()

Retrieves the name of the host associated with the specified single sign-on token handle.

Syntax

```
#include "am_sso.h"
AM_EXPORT const char *
am_sso_get_host(const am_sso_token_handle_t sso_token);
```

Parameters

This function takes the following parameter:

`sso_token` Pointer to a `am_sso_token_handle_t` type.

Returns

This function returns the host name as defined in the `Host` property, or `NULL` if the `Host` property is not set or does not have a value.

am_sso_get_idle_time

Retrieves the idle time associated with the specified single sign-on token handle.

Syntax

```
#include "am_sso.h"
AM_EXPORT time_t
am_sso_get_idle_time(const am_sso_token_handle_t sso_token_handle);
```

Parameters

This function takes the following parameter:

`sso_token_handle`	Pointer to a `am_sso_token_handle_t` type.

Returns

This function returns the idle time for this session in seconds, or the standard `time_t` data structure in the form `(time_t) -1` if the token is invalid or some type of error occurs. Detailed error information is logged.

am_sso_get_max_idle_time()

Retrieves the maximum idle time associated with the specified single sign-on token handle.

Syntax

```
#include "am_sso.h"
AM_EXPORT time_t
am_sso_get_max_idle_time(const am_sso_token_handle_t sso_token);
```

Parameters

This function takes the following parameters:

`sso_token`	Pointer to a `am_sso_token_handle_t` type.

Returns

This function returns the maximum idle time for this session in seconds, or the standard `time_t` data structure in the form `(time_t) -1` if some type of error occurs.

am_sso_get_max_session_time()

Retrieves the maximum session time associated with the specified single sign-on token handle.

Syntax

```
#include "am_sso.h"
AM_EXPORT time_t
am_sso_get_max_session_time(const am_sso_token_handle_t sso_token);
```

Parameters

This function takes the following parameter:

`sso_token` Pointer to a `am_sso_token_handle_t` type.

Returns

This function returns the maximum session time for this session in seconds, or the standard `time_t` data structure in the form `(time_t) -1` if some type of error occurs.

am_sso_get_principal()

Retrieves the principal (user) associated with the specified single sign-on token handle.

Syntax

```
#include "am_sso.h"
AM_EXPORT const char *
am_sso_get_principal(const am_sso_token_handle_t sso_token);
```

This function takes the following parameter:

`sso_token` Pointer to a `am_sso_token_handle_t` type.

Returns

This function returns the principal (user) of the specified session, or `NULL` if the handle is invalid or any other error occurred.

am_sso_get_principal_set()

Retrieves a set of principals associated with the specified single sign-on token handle.

Syntax

```
#include "am_sso.h"
AM_EXPORT am_string_set_t *
am_sso_get_principal_set(const am_sso_token_handle_t sso_token);
```

Parameters

This function takes the following parameter:

`sso_token`	Pointer to a `am_sso_token_handle_t` type.

Returns

This function returns the `am_string_set_t` type (defined in the `<am_string_set.h>` header file) that points to the set of principals associated with the specified single sign-on token handle. It returns `NULL` if the applicable property is not set or has no value.

am_sso_get_property()

Retrieves the value of a property associated with the specified single sign-on token handle.

Syntax

```
#include "am_sso.h"
AM_EXPORT const char *
am_sso_get_property(const am_sso_token_handle_t sso_token,
                    const char *property_key,
                    boolean_t check_if_session_valid);
```

Parameters

This function takes the following parameters:

`sso_token`	Pointer to a `am_sso_token_handle_t` type.
`property_key`	Pointer to the name of the desired property.
`check_if_session_valid`	Takes a value based on the `boolean_t` (defined in the `<am_types.h>` header file) that specifies if the function should check first if the session is valid. If the session is invalid, `NULL` will always be returned.

Returns

This function returns a pointer to the value of the property, or `NULL` if the property is not set or does not have a value.

am_sso_get_sso_token_id()

Retrieves the `SSOTokenID` associated with the specified single sign-on token handle.

Syntax

```
#include "am_sso.h"
AM_EXPORT const char *
am_sso_get_sso_token_id(const am_sso_token_handle_t sso_token_handle);
```

Parameters

This function takes the following parameters:

`sso_token_handle`	Pointer to a `am_sso_token_handle_t` type.

Returns

This function returns a pointer to the `SSOTokenID`, or `NULL` if `sso_token_handle` is invalid or any other error occurred.

am_sso_get_time_left()

Retrieves the time left in the session associated with the specified single sign-on token handle.

Syntax

```
#include "am_sso.h"
AM_EXPORT time_t
am_sso_get_time_left(const am_sso_token_handle_t sso_token_handle);
```

Parameters

This function takes the following parameters:

`sso_token_handle`	Pointer to a `am_sso_token_handle_t` type.

Returns

This function returns the time left on this session in seconds, or the standard `time_t` data structure in the form `(time_t) -1` if the token is invalid or some other type of error occurs. Detailed error information is logged.

am_sso_init()

Initializes the data structures, allowing communication with the Session Service.

Details

am_sso_init() takes as input a properties file that contains name/value pairs, and returns status on the success or failure of the initialization. This call must be made before calling any other am_sso_* functions. See "Single Sign-on Properties" on page 65 for more information.

Syntax

```
#include "am_sso.h"
AM_EXPORT am_status_t
am_sso_init(am_properties_t property_map);
```

Parameters

This function takes the following parameter:

property_map   Pointer to the am_properties_t structure used to initialize the Session Service.

Note – See "am_properties_t" on page 131 for more information.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS   If the initialization was successful.

AM_*   If any error occurs, the type of error indicated by the status value.

am_sso_invalidate_token()

Invalidates or destroys the session on Access Manager associated with the single sign-on token handle.

Details

am_sso_invalidate_token() does not free the sso_token_handle parameter. You must call am_sso_destroy_sso_token_handle() to free the memory for the handle itself.

Syntax

```
#include "am_sso.h"
AM_EXPORT am_status_t
am_sso_invalidate_token(const am_sso_token_handle_t sso_token_handle);
```

Parameters

This function takes the following parameter:

`sso_token_handle`	Pointer to a `am_sso_token_handle_t` type.

Note – If successful, the single sign-on token handle will have an invalid state after this call.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the <am_types.h> header file):

`AM_SUCCESS`	If session was successfully invalidated.
`AM_INVALID_ARGUMENT`	If the `sso_token_handle` parameter is `NULL`.
`AM_SERVICE_NOT_INITIALIZED`	If the Session Service was not initialized with `am_sso_init()`.
`AM_SERVICE_NOT_AVAILABLE`	If server returned service not available.
`AM_HTTP_ERROR`	If an HTTP error was encountered while communicating with Access Manager.
`AM_ERROR_PARSING_XML`	If an error occurred while parsing XML from Access Manager.
`AM_ACCESS_DENIED`	If access was denied while communicating with Access Manager.
`AM_FAILURE`	If any other error occurred.

am_sso_is_valid_token()

Checks if the `SSOToken` associated with the specified single sign-on token handle is valid.

Details

`am_sso_is_valid_token()` looks in the passed `sso_token_handle` to check for validity. It does *not* go to Access Manager.

Syntax

```
#include "am_sso.h"
AM_EXPORT boolean_t
am_sso_is_valid_token(const am_sso_token_handle_t sso_token_handle);
```

Parameters

This function takes the following parameter:

`sso_token_handle`	Pointer to a `am_sso_token_handle_t` type.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the `<am_types.h>` header file):

`B_TRUE`	If the single sign-on token is valid.
`B_FALSE`	If the single sign-on token is invalid or any other error occurred.

am_sso_refresh_token()

Refreshes the session information in the `SSOToken` associated with the specified single sign-on token handle.

Details

`am_sso_refresh_token()` goes to Access Manager to retrieve the latest session information with which to update the `SSOToken`. This is similar in functionality to `am_sso_validate_token()` however, `am_sso_refresh_token()` also refreshes the last access time of the session.

Syntax

```
#include "am_sso.h"
AM_EXPORT am_status_t
am_sso_refresh_token(const am_sso_token_handle_t sso_token_handle);
```

Parameters

This function takes the following parameter:

`sso_token_handle`	Pointer to a `am_sso_token_handle_t` type.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the token was refreshed with no errors.
`AM_INVALID_ARGUMENT`	If the `sso_token_handle` parameter is invalid.
`AM_SERVICE_NOT_INITIALIZED`	If the Session Service was not initialized with `am_sso_init()`.

AM_SERVICE_NOT_AVAILABLE	If server returned service not available.
AM_HTTP_ERROR	If an HTTP error was encountered while communicating with Access Manager.
AM_ERROR_PARSING_XML	If an error occurred while parsing XML from Access Manager.
AM_ACCESS_DENIED	If access was denied while communicating with Access Manager.
AM_SESSION_FAILURE	If the session validation failed.
AM_FAILURE	If any other error occurred.

am_sso_remove_listener()

Removes a single sign-on token listener.

Details

If am_sso_add_listener() was called more than once for the same listener function, all instances of the listener function will be removed.

Syntax

```
#include "am_sso.h"
AM_EXPORT am_status_t
am_sso_remove_listener(const am_sso_token_listener_func_t listener);
```

Parameters

This function takes the following parameter:

listener	The listener as described in "am_sso_token_listener_func_t" on page 73.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the listener was successfully removed.
AM_INVALID_ARGUMENT	If the listener is NULL.
AM_NOT_FOUND	If listener was not found.
AM_FAILURE	If any other error occurred.

am_sso_remove_sso_token_listener()

Removes a single sign-on token listener associated with the specified single sign-on token handle.

Details

If am_sso_add_listener() was called more than once for the same listener function, all instances of the listener function will be removed.

Syntax

```
#include "am_sso.h"
AM_EXPORT am_status_t
am_sso_remove_sso_token_listener(const am_sso_token_handle_t sso_token_handle,
                                 const am_sso_token_listener_func_t listener);
```

Parameters

This function takes the following parameters:

sso_token_handle	Pointer to a am_sso_token_handle_t type.
listener	The listener as described in "am_sso_token_listener_func_t" on page 73.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the listener was successfully removed.
AM_INVALID_ARGUMENT	If the listener is NULL.
AM_NOT_FOUND	If listener was not found.
AM_FAILURE	If any other error occurred.

am_sso_set_property()

Sets a property and its value in the SSOToken associated with the specified single sign-on token handle.

Details

The single sign-on token handle for this SSOToken was obtained before this call and thus will not include the new property. You must call am_sso_validate_token() to update the handle.

Syntax

```
#include "am_sso.h"
AM_EXPORT am_status_t
am_sso_set_property(am_sso_token_handle_t sso_token_handle,
                    const char *name,
                    const char *value);
```

Parameters

This function takes the following parameters:

`sso_token_handle` Pointer to a `am_sso_token_handle_t` type.

`name` Pointer to the name of the property.

> **Caution –** If the specified property is protected by Access Manager, `am_sso_set_property()` will return success, but the value given will not be set.

`value` Pointer to the value for the specified property.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS` If the property was successfully set.

`AM_INVALID_ARGUMENT` If the `sso_token_handle` is invalid.

`AM_FAILURE` If any other error occurred.

`am_sso_validate_token()`

Updates the session information in the `SSOToken` associated with the specified single sign-on token handle.

Details

This call will go to Access Manager to retrieve the latest session information. The `sso_token_handle` is updated if the return status is either `AM_SUCCESS` or `AM_INVALID_SESSION`. This is different from `am_sso_refresh_token()` in that it does *not* update the last access time on the server.

Syntax

```
#include "am_sso.h"
AM_EXPORT am_status_t
am_sso_validate_token(const am_sso_token_handle_t sso_token_handle);
```

Parameters

This function takes the following parameter:

`sso_token_handle`	Pointer to a `am_sso_token_handle_t` type.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the <am_types.h> header file):

`AM_SUCCESS`	If single sign-on token is valid. The information is updated.
`AM_INVALID_SESSION`	If the session is invalid. The information is updated.
`AM_INVALID_ARGUMENT`	If the input parameter is invalid.
`AM_SERVICE_NOT_INITIALIZED`	If Session Service is not initialized.
`AM_SERVICE_NOT_AVAILABLE`	If Access Manager returned service not available.
`AM_HTTP_ERROR`	If HTTP error encountered while communicating with Access Manager.
`AM_ERROR_PARSING_XML`	If error parsing XML from Access Manager.
`AM_ACCESS_DENIED`	If access is denied while communicating with Access Manager.
`AM_FAILURE`	If any other error occurred.

CHAPTER 5

Logging Data Types and Functions

Sun™ Java System Access Manager contains public data types and functions you can use for logging on the local system or on Sun Java™ System Access Manager. Reference summaries include a short description, syntax, parameters and returns. The code is contained in the <am_log.h> header file (located in the /*AccessManager-base*/`SUNWam/include` directory). The sample source `am_log_test.c` (located in the /*AccessManager-base*/`SUNWam/samples/csdk` directory) demonstrates the basic usage of the logging API to record information such as user activity, traffic patterns and authorization violations. This chapter contains the following sections:

- "The Logging API for C" on page 93
- "Logging Data Types" on page 94
- "Logging Functions" on page 95

The Logging API for C

The logging API is designed to allow C applications to produce messages of interest and write them to the Access Manager logs. When some type of event occurs in an external application, the application code first determines if the *logging module* (a file created for messages usually relevant to a specific function or feature) to which the event is relevant has a level high enough to log the event. (A *level* specifies importance and defines the amount of detail that will be logged.) If the determination is affirmative, a log message is generated and a *log record* created in the relevant logging module. Information in the log record can be updated as necessary. The following notes regard the logging API for C functionality:

- The `am_log_init()` function by the application must be called before using any other `am_log_*` interfaces. If either the SSO, authentication, or policy initialization functions (`am_sso_init()`, `am_auth_init()`, or `am_policy_init()`) are called, `am_log_init()` does not need to be called as each of the three aforementioned functions call `am_log_init()` internally.
- The **`am_log_record_`*** interfaces can be used to set or update information in the log record. They include:

  - "`am_log_record_add_loginfo()`" on page 101
  - "`am_log_record_create()`" on page 101

  - "am_log_record_destroy()" on page 103
  - "am_log_record_populate()" on page 103
  - "am_log_record_set_log_level()" on page 104
  - "am_log_record_set_log_message()" on page 105
  - "am_log_record_set_loginfo_props()" on page 105

- The following are convenience functions that provide simplified access to existing log records. They include:

  - "am_log_record_set_log_level()" on page 104
  - "am_log_record_set_log_message()" on page 105

Logging Data Types

The logging data types defined in <am_log.h> are:

- "am_log_record_t" on page 94
- "am_log_module_id_t" on page 94

am_log_record_t

Represents the information and message values to be recorded.

Note – See Chapter 6, "Access Manager Logging and Java Enterprise System Monitoring Framework," in *Sun Java System Access Manager 7.1 Technical Overview* for information regarding events that are logged and the log fields to which they are written.

Syntax

```
#include "am_log.h"
typedef struct am_log_record *am_log_record_t;
```

Members

am_log_record is an opaque structure with no accessible members.

am_log_module_id_t

Represents the identifier for a logging module.

Note – See "am_log_add_module()" on page 95 for information on logging modules.

Syntax

```
#include "am_log.h"
typedef unsigned int am_log_module_id_t;
```

Members

`am_log_module_id_t` has no members.

Logging Functions

The logging functions defined in `<am_log.h>` are:

`am_log_add_module()`

Adds a new logging file (for a specific function or feature) to the Access Manager Logging Service.

Details

The currently used *module* file names are:

- AuthService
- NamingService
- PolicyService
- SessionService
- PolicyEngine

- ServiceEngine
- Notification
- PolicyAgent
- RemoteLog
- all

Syntax

```
#include "am_log.h"
AM_EXPORT am_status_t
am_log_add_module(const char *name,
                  am_log_module_id_t *id_ptr);
```

Parameters

This function takes the following parameters:

`name`	Pointer to the name associated with the new module.
`id_ptr`	Pointer to the location where the identifier for the logging module is stored.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

Note – If a module of the same name already exists, the module identifier of the existing module is returned.

`AM_SUCCESS`	If the addition was successful.
`AM_INVALID_ARGUMENT`	If `name` or `id_ptr` is `NULL`.
`AM_NSPR_ERROR`	If unable to initialize the logging framework.
`AM_NO_MEMORY`	If unable to allocate memory for the new module.
`AM_FAILURE`	If any other error is detected.

`am_log_flush_remote_log()`

Flushes all the log records in the Access Manager log buffer.

Syntax

```
#include "am_log.h"
AM_EXPORT am_status_t
am_log_flush_remote_log();
```

Parameters

This function takes no parameters:

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the flush was successful.
`AM_*`	The appropriate code based on the error.

`am_log_init()`

Initializes the Logging Service.

Details

`am_log_init()` writes events to the logs on the Access Manager server.

Syntax

```
#include "am_log.h"
AM_EXPORT am_status_t
am_log_init(const am_properties_t log_init_params);
```

Parameters

This function takes the following parameter:

`log_init_params`	Properties used to initialize the Logging Service.

Note – See “`am_properties_t`” on page 131 for more information.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If log initialization is successful.
`AM_*`	If any error occurs, the type of error indicated by the status value.

am_log_is_level_enabled()

Checks whether an event at the specified level intended for the specified logging module should generate a logging message.

Details

If the level of the event is not equal to or higher than the level of the logging module, a logging message will not be generated.

Syntax

```
#include "am_log.h"
AM_EXPORT boolean_t
am_log_is_level_enabled(am_log_module_id_t moduleID,
                        am_log_level_t level);
```

Parameters

This function takes the following parameters:

`moduleID`	The identifier of the logging module.
`level`	The level of the event. Possible values are defined in the following `am_log_level_t` enumeration. The default value is `AM_LOG_INFO`.

```
typedef enum am_log_level {
    AM_LOG_ALWAYS = -1, /* always logged */
    AM_LOG_NONE,    /* never logged, typically used to turn off a module */
    AM_LOG_ERROR,    /* used for error messages */
    AM_LOG_WARNING,    /* used for warning messages */
    AM_LOG_INFO,    /* used for informational messages */
    AM_LOG_DEBUG,    /* used for debug messages */
    AM_LOG_MAX_DEBUG,    /* used for more detailed debug messages */
    AM_LOG_AUTH_REMOTE = 128, /* logged deny and/or allow */
    AM_LOG_AUTH_LOCAL = 256
} am_log_level_t;
```

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the standard `<types.h>` header file):

Note – The code used is dependent on the server operating system.

`!0`	If a message will be generated.

0 If a message will not be generated.

am_log_log()

Produces a logging message from the specified event string.

Details

When using am_log_log(), consider the following:

- The message is produced only if the level defined for the specified module is greater than or equal to the level defined for the message. See "am_log_is_level_enabled()" on page 98.
- am_log_log() directly enumerates arguments for the format string. See "am_log_vlog()" on page 109 for another method.

Syntax

```
#include "am_log.h"
AM_EXPORT boolean_t
am_log_log(am_log_module_id_t moduleID,
           am_log_level_t level,
           const char *format, ...);
```

Parameters

This function takes the following parameters:

moduleID	The identifier of the Access Manager logging module to which the message is relevant.
level	The level of the message. Each message has an associated level that defines the amount of detail that will be logged. Possible values are defined in the am_log_level_t enumeration. The default value is AM_LOG_INFO.

```
typedef enum am_log_level {
    AM_LOG_ALWAYS = -1, /* always logged */
    AM_LOG_NONE,    /* never logged, typically used to turn off a module */
    AM_LOG_ERROR,    /* used for error messages */
    AM_LOG_WARNING,    /* used for warning messages */
    AM_LOG_INFO,    /* used for informational messages */
    AM_LOG_DEBUG,    /* used for debug messages */
    AM_LOG_MAX_DEBUG,    /* used for more detailed debug messages */
    AM_LOG_AUTH_REMOTE = 128, /* logged deny and/or allow */
    AM_LOG_AUTH_LOCAL = 256
} am_log_level_t;
```

format	Pointer to a printf-style character string detailing the event.

Note – The set of additional arguments needed by `format` are either enumerated directly or passed using the standard `va_list` mechanism as appropriate to the call. See "`am_log_vlog()`" on page 109.

Returns

This function returns one of the values of the `boolean_t` enumeration (defined in the standard `<types.h>` header file):

Note – The code used is dependent on the server operating system.

`!0` If the message is logged.

`0` If the message will not be logged.

`am_log_log_record()`

Writes the given log record to the specified logging module.

Syntax

```
#include "am_log.h"
AM_EXPORT am_status_t
am_log_log_record(am_log_record_t record,
                  const char *log_name,
                  const char *logged_by_token_id);
```

Parameters

This function takes the following parameters:

`record` The log record pointer.

`log_name` Pointer to the name of the logging module to which the log record will be written.

`logged_by_token_id` Pointer to a valid `SSOTokenID` identifying the user to whom the log record applies.

Note – This is required to access the Logging Service on Access Manager.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If log record is written.
`AM_*`	If any error occurs, the type of error indicated by the status value.

am_log_record_add_loginfo()

Updates a log record with additional information.

Syntax

```
#include "am_log.h"
AM_EXPORT am_status_t
am_log_record_add_loginfo(am_log_record_t record,
                          const char *key,
                          const char *value);
```

Parameters

This function takes the following parameters:

`record` A log record pointer.

`key` Pointer to the log field being updated.

> **Note** – See Chapter 6, "Access Manager Logging and Java Enterprise System Monitoring Framework," in *Sun Java System Access Manager 7.1 Technical Overview* for information regarding events that are logged and the log fields to which they are written.

`value` Pointer to the value with which the log field will be modified.

Returns

This function returns one of the values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file).

am_log_record_create()

Instantiates a log record, initializing it with a message and the message's corresponding level.

Syntax

```
#include "am_log.h"
 AM_EXPORT am_status_t
am_log_record_create(am_log_record_t *record_ptr,
                     am_log_record_log_level_t log_level,
                     const char *message);
```

Parameters

This function takes the following parameters:

`record_ptr` Pointer to a log record pointer.

`log_level` The level with which the log record will be instantiated. Each log record has an associated level that defines its relative importance and urgency. Possible values are defined in the `am_log_record_log_level_t` enumeration.

```
typedef enum am_log_record_log_level {

    /* Log Level as defined by JDK 1.4 */

    AM_LOG_LEVEL_SEVERE = 1000,
    AM_LOG_LEVEL_WARNING = 900,
    AM_LOG_LEVEL_INFORMATION = 800,
    AM_LOG_LEVEL_CONFIG = 700,
    AM_LOG_LEVEL_FINE = 500,
    AM_LOG_LEVEL_FINER = 400,
    AM_LOG_LEVEL_FINEST = 300,

    /* Log Levels defined by Access Manager */

    AM_LOG_LEVEL_SECURITY = 950,
    AM_LOG_LEVEL_CATASTROPHE = 850,
    AM_LOG_LEVEL_MISCONF = 750,
    AM_LOG_LEVEL_FAILURE = 650,
    AM_LOG_LEVEL_WARN = 550,
    AM_LOG_LEVEL_INFO = 450,
    AM_LOG_LEVEL_DEBUG = 350,
    AM_LOG_LEVEL_ALL = 250

} am_log_record_log_level_t;
```

`message` Pointer to the log message to be written to the log record.

Returns

This function returns one of the values of the `am_status_t` enumeration (defined in the <am_types.h> header file).

am_log_record_destroy()

Destroys the specified log record.

Syntax

```
#include "am_log.h"
AM_EXPORT am_status_t
am_log_record_destroy(am_log_record_t record);
```

Parameters

This function takes the following parameter:

`record`	A log record pointer.

Returns

This function returns one of the values of the `am_status_t` enumeration (defined in the <am_types.h> header file).

am_log_record_populate()

Updates a log record with the user's session identifier (also known as an `SSOTokenID`).

Details

See "Single Sign-on Token Handles" on page 68 for more information.

Syntax

```
#include "am_log.h"
AM_EXPORT am_status_t
am_log_record_populate(am_log_record_t record,
                       const char *user_token_id);
```

Parameters

This function takes the following parameters:

`record`	A log record pointer.
`user_token_id`	Pointer to a valid session identifier (also known as an `SSOTokenID`).

Returns

This function returns one of the values of the `am_status_t` enumeration (defined in the <am_types.h> header file).

am_log_record_set_log_level()

Sets the level for the specified log record.

Syntax

```
#include "am_log.h"
AM_EXPORT am_status_t
am_log_record_set_log_level(am_log_record_t record,
                            am_log_record_log_level_t log_level);
```

Parameters

This function takes the following parameters:

`record` A log record pointer.

`log_level` The level to which the log record will be set. Each log record has an associated level that defines its relative importance and urgency. Possible values are defined in the `am_log_record_log_level_t` enumeration.

```
typedef enum am_log_record_log_level {

    /* Log Level as defined by JDK 1.4 */

    AM_LOG_LEVEL_SEVERE = 1000,
    AM_LOG_LEVEL_WARNING = 900,
    AM_LOG_LEVEL_INFORMATION = 800,
    AM_LOG_LEVEL_CONFIG = 700,
    AM_LOG_LEVEL_FINE = 500,
    AM_LOG_LEVEL_FINER = 400,
    AM_LOG_LEVEL_FINEST = 300,

    /* Log Levels defined by Access Manager */

    AM_LOG_LEVEL_SECURITY = 950,
    AM_LOG_LEVEL_CATASTROPHE = 850,
    AM_LOG_LEVEL_MISCONF = 750,
    AM_LOG_LEVEL_FAILURE = 650,
    AM_LOG_LEVEL_WARN = 550,
    AM_LOG_LEVEL_INFO = 450,
    AM_LOG_LEVEL_DEBUG = 350,
```

```
        AM_LOG_LEVEL_ALL = 250

    } am_log_record_log_level_t;
```

Returns

This function returns one of the values of the am_status_t enumeration (defined in the <am_types.h> header file).

am_log_record_set_log_message()

Sets the log message to the log record before localization and formatting.

Syntax

```
#include "am_log.h"
AM_EXPORT am_status_t
am_log_record_set_log_message(am_log_record_t record,
                              const char *message);
```

Parameters

This function takes the following parameters:

record	A log record pointer.
message	Pointer to the log message to be written to the log record.

Returns

This function returns one of the values of the am_status_t enumeration (defined in the <am_types.h> header file).

am_log_record_set_loginfo_props()

Updates the specified log record with additional information.

Details

log_info is expected to have the information formatted as key/value pairs in a properties map. Delete the am_properties_t pointer only when finished with the SDK. See "am_properties_t" on page 131 for more information.

Syntax

```
#include "am_log.h"
AM_EXPORT am_status_t
am_log_record_set_loginfo_props(am_log_record_t record,
                                am_properties_t log_info);
```

Parameters

This function takes the following parameters:

`record`	A log record pointer.
`log_info`	Pointer to the properties that contain the information to be set in the log record.

Returns

This function returns one of the values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file).

am_log_set_levels_from_string()

Sets the level for the logging modules listed in a specified string.

Details

The format of the string must be:

ModuleName`[:`*Level*`][,`*ModuleName*`[:`*Level*`]]*`

Optional spaces may occur before and after any commas. The comma, brackets and asterisk in the second term signifies that it can occur 0 or more times.

Syntax

```
#include "am_log.h"
AM_EXPORT am_status_t
am_log_set_levels_from_string(const char *module_level_string);
```

Parameters

This function takes the following parameter:

`module_level_string`	Pointer to the string containing the list of modules and the respective levels.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the levels were successfully set.
`AM_INVALID_ARGUMENT`	If `module_level_string` is `NULL`.
`AM_FAILURE`	If any other error is detected.

am_log_set_log_file()

Sets the name of the file to use for logging.

Syntax

```
#include "am_log.h"
AM_EXPORT am_status_t
am_log_set_log_file(const char *name);
```

Parameters

This function takes the following parameter:

`name`	Pointer to the name of the file to which log records are recorded.

Note – If `NULL` or empty, logging messages are sent to the `stderr` stream.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the logging file is successfully set.
`AM_NO_MEMORY`	If unable to allocate memory for internal data structures.
`AM_FAILURE`	If an error of any type occurs.

am_log_set_module_level()

Sets the level for the specified logging module.

Syntax

```
#include "am_log.h"
AM_EXPORT am_log_level_t
am_log_set_module_level(am_log_module_id_t moduleID,
                        am_log_level_t level);
```

Parameters

This function takes the following parameters:

`moduleID`	The identifier of the logging module.
`level`	The level to which the logging module will be set. Each module has an associated level that defines the amount of detail that will be logged. Possible values are defined in the following enumeration. The default value is `AM_LOG_INFO`.

```
typedef enum am_log_level {
    AM_LOG_ALWAYS = -1, /* always logged */
    AM_LOG_NONE,    /* never logged, typically used to turn off a module */
    AM_LOG_ERROR,    /* used for error messages */
    AM_LOG_WARNING,    /* used for warning messages */
    AM_LOG_INFO,    /* used for informational messages */
    AM_LOG_DEBUG,    /* used for debug messages */
    AM_LOG_MAX_DEBUG,    /* used for more detailed debug messages */
    AM_LOG_AUTH_REMOTE = 128, /* logged deny and/or allow */
    AM_LOG_AUTH_LOCAL = 256
} am_log_level_t;
```

Returns

This function returns `am_log_level_t` with one of the following values:

The previous logging level of the module.	If the logging level is set properly.
`LOG_NONE`	If the specified module is invalid.

am_log_set_remote_info()

Initializes the remote log service.

Details

This must be called before `am_log_log()` with `AM_LOG_REMOTE_MODULE` as the log module. Initialization is done only once. Subsequently, only remote logging calls are done.

Syntax

```
#include "am_log.h"
AM_EXPORT am_status_t
am_log_set_remote_info(const char *rem_log_url,
                       const char *sso_token_id,
                       const char *rem_log_name,
                       const am_properties_t log_props);
```

Parameters

This function takes the following parameters:

`rem_log_url`	Pointer to the URL of the Access Manager Logging Service being used for the remote logging.
`sso_token_id`	Pointer to a valid `SSOTokenID` identifying the user to whom the log record applies.
`rem_log_name`	Pointer to the logging module (file) to which log records are written.
`log_props`	Pointer to the properties that contain the information to initialize the Access Manager Logging Service.

Note – `log_props` is expected to have the information formatted as a properties map in key/value pairs. Delete the `am_properties_t` pointer only when finished with the SDK. See "`am_properties_t`" on page 131 for more information.

Returns

This function returns one of the values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file).

`am_log_vlog()`

Logs a message for the specified module at the given level.

Details

When using `am_log_vlog()`, consider the following:

- The message is produced only if the level defined for the specified module is greater than or equal to the level defined for the message. See "`am_log_is_level_enabled()`" on page 98.
- `am_log_vlog()` passes the standard `va_list` as an argument for the `format` string. See "`am_log_log()`" on page 99 for another method.

Syntax

```
#include "am_log.h"
AM_EXPORT boolean_t
am_log_vlog(am_log_module_id_t moduleID,
            am_log_level_t level,
            const char *format,
            va_list args);
```

Parameters

This function takes the following parameters:

`moduleID` The identifier of the Access Manager logging module to which the message is relevant.

`level` The level of the message. Each message has an associated level that defines the amount of detail that will be logged. Possible values are defined in the `am_log_level_t` enumeration. The default value is `AM_LOG_INFO`.

```
typedef enum am_log_level {
    AM_LOG_ALWAYS = -1, /* always logged */
    AM_LOG_NONE,    /* never logged, typically used to turn off a module */
    AM_LOG_ERROR,    /* used for error messages */
    AM_LOG_WARNING,    /* used for warning messages */
    AM_LOG_INFO,    /* used for informational messages */
    AM_LOG_DEBUG,    /* used for debug messages */
    AM_LOG_MAX_DEBUG,    /* used for more detailed debug messages */
    AM_LOG_AUTH_REMOTE = 128, /* logged deny and/or allow */
    AM_LOG_AUTH_LOCAL = 256
} am_log_level_t;
```

`format` Pointer to a `printf-style` character string.

Note – The set of addition arguments needed by `format` are either enumerated directly or passed using the standard `va_list` mechanism as appropriate to the call.

`va_list` A void pointer interpreted as an argument list. `va_list` is the type of the void pointer passed to a function that accepts a pointer to a list of arguments.

Note – The set of additional arguments needed by `format` are either enumerated directly or passed using the standard `va_list` mechanism as appropriate to the call. See “`am_log_log()`” on page 99.

Returns

This function returns one of the values of the `boolean_t` enumeration (defined in the standard `<types.h>` header file):

Note – The code used is dependent on the server operating system. See <am_types.h> for more details.

`!0` If the message can be logged.

`0` If the message will not be logged.

CHAPTER 6

Mapping Data Types and Functions

Sun™ Java System Access Manager contains public data types and functions you can use for creating, destroying, and manipulating map objects. Reference summaries include a short description, syntax, parameters and returns. The code is contained in the <am_map.h> header file (located in the `/`*AccessManager-base*`/SUNWam/include` directory). The sample source `am_policy_test.c` (located in the `/`*AccessManager-base*`/SUNWam/samples/csdk` directory) demonstrates the basic usage of some of the basic mapping functions. This chapter contains the following sections:

The Mapping API for C

A *map* is an object that associates a *key* to a *value*. One key/value pair is an *entry* in the map. Maps are used by the policy API for C to return policy decision results from the Policy Service. They are also used to pass any environment variables to the Policy Service for evaluation.

Mapping Data Types

The mapping types defined in <am_map.h> are:

am_map_t

Pointer to a map object consisting of key/value entry mappings.

Syntax

```
#include "am_map.h"
typedef struct am_map *am_map_t;
```

Members

`am_map` is an opaque structure with no accessible members.

Memory Concerns

Free the allocated structure by calling `am_map_destroy()`. See "`am_map_destroy()`" on page 117.

`am_map_entry_iter_t`

Pointer to an iterator for the entries in a map object.

Syntax

```
#include "am_map.h"
typedef struct am_map_entry_iter *am_map_entry_iter_t;
```

Members

`am_map_entry_iter` is an opaque structure with no accessible members.

`am_map_value_iter_t`

Pointer to an iterator for the values in a map object associated with a specified key.

Syntax

```
#include "am_map.h"
typedef struct am_map_value_iter *am_map_value_iter_t;
```

Members

`am_map_value_iter` is an opaque structure with no accessible members.

Mapping Functions

The mapping functions defined in <am_map.h> are:

am_map_clear()

Erases all of the entries in the specified map object.

Syntax

```
#include "am_map.h"
AM_EXPORT am_status_t
am_map_clear(am_map_t map);
```

Parameters

This function takes the following parameter:

`map` The map object.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the <am_types.h> header file):

`AM_SUCCESS`	If the entries were successfully erased.
`AM_INVALID_ARGUMENT`	If the map argument is `NULL`.

`am_map_copy()`

Makes a copy of the specified map object.

Details

`am_map_copy()` creates a new instance of a `am_map_t`, copies all the elements from the specified `source_map` into it, and assigns to the new instance a pointer. It does not alter the contents of the original map object.

Syntax

```
#include "am_map.h"
AM_EXPORT am_status_t
am_map_copy(am_map_t source_map,
            am_map_t *map_ptr);
```

Parameters

This function takes the following parameters:

`source_map`	The specified map object. It may be `NULL`.
`map_ptr`	Pointer to the location of the new map object copy.

Caution – Be sure not to pass `map_ptr` as a valid `am_map` structure as the reference will be lost.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the <am_types.h> header file):

`AM_SUCCESS`	If the map object was successfully copied.
`AM_NO_MEMORY`	If unable to allocate memory for the new map object.
`AM_INVALID_ARGUMENT`	If the `source_map` or `map_ptr` argument is `NULL`.

Memory Concerns

The caller must destroy `map_ptr` after usage by calling `am_map_destroy()`.

am_map_create()

Creates a new, empty map object.

Details

am_map_create() creates an instance of a am_map_t and returns a pointer back to the caller.

Syntax

```
#include "am_map.h"
AM_EXPORT am_status_t
am_map_create(am_map_t *map_ptr);
```

Parameters

This function takes the following parameter:

map_ptr Pointer specifying the location of the new map object.

Caution – Be sure not to pass map_ptr as a valid am_map structure as the reference will be lost.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the map object was successfully created.
AM_NO_MEMORY	If unable to allocate memory for the new map object.
AM_INVALID_ARGUMENT	If the map_ptr argument is NULL.

am_map_destroy()

Destroys the specified map object.

Syntax

```
#include "am_map.h"
AM_EXPORT void
am_map_destroy(am_map_t map);
```

Parameters

This function takes the following parameter:

`map` The specified map object. It may be `NULL`.

Caution – Be sure not to pass `map` as a valid `am_map` structure as the reference will be lost.

Returns

This function does not return a value.

Memory Concerns

The pointer to the specified map object can not be freed before calling `am_map_destroy()`. This includes erroneously calling the system `free(void *)` function.

am_map_entry_iter_destroy()

Destroys the specified entry iterator.

Syntax

```
#include "am_map.h"
AM_EXPORT void
am_map_entry_iter_destroy(am_map_entry_iter_t entry_iter);
```

Parameters

This function takes the following parameter:

`entry_iter` The specified entry iterator. It may be `NULL`.

Returns

This function does not return a value.

am_map_entry_iter_get_first_value()

Returns the first value assigned to the entry currently being referenced by the specified entry iterator.

Syntax

```
#include "am_map.h"
AM_EXPORT const char *
am_map_entry_iter_get_first_value(am_map_entry_iter_t entry_iter);
```

Parameters

This function takes the following parameter:

`entry_iter`	The specified entry iterator. It may be `NULL`.

Returns

This function returns one of the following:

`char *`	Returns the first associated value of the specified key. The order of insertion into the map does not guarantee the value returned.
`NULL`	If the specified iterator is `NULL`, does not reference a valid entry, or the entry does not have any associated values.

Memory Concerns

`am_map_entry_iter_get_first_value()` destroys the `am_map_entry_iter_t` passed to it. Because of this, don't call this function more than once on the same `am_map_entry_iter_t`.

am_map_entry_iter_get_key()

Returns the key assigned to the entry currently being referenced by the specified entry iterator.

Syntax

```
#include "am_map.h"
AM_EXPORT const char *
am_map_entry_iter_get_key(am_map_entry_iter_t entry_iter);
```

Parameters

This function takes the following parameters:

`entry_iter`	The specified entry iterator.

Returns

This function returns one of the following values:

`char *`	Returns the key.

Note – Caller must not modify or free the return value.

`NULL`	If the specified key iterator is `NULL` or does not reference a valid entry.

am_map_entry_iter_get_values()

Returns a value iterator that can be used to sequence through the values assigned to the entry currently being referenced by the specified entry iterator.

Syntax

```
#include "am_map.h"
AM_EXPORT am_status_t
am_map_entry_iter_get_values(am_map_entry_iter_t entry_iter,
                             am_map_value_iter_t *value_iter_ptr);
```

Parameters

This function takes the following parameters:

`entry_iter`	The specified entry iterator.
`value_iter_ptr`	Pointer specifying the location of the value iterator.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the <am_types.h> header file):

`AM_SUCCESS`	If no error was detected.
`AM_NO_MEMORY`	If unable to allocate memory for the key iterator.
`AM_INVALID_ARGUMENT`	If the `value_iter_ptr` argument is `NULL`.

Note – If `value_iter_ptr` is not `NULL` and an error is returned, the location that it references will be set to `NULL`.

`AM_NOT_FOUND`	If the `entry_iter` argument is `NULL` or does not reference a valid entry.

Memory Concerns

After using `am_map_value_iter_t`, the caller must call `am_map_value_iter_destroy()`.

am_map_entry_iter_is_entry_valid()

Determines if the entry currently being referenced by the specified entry iterator is valid.

Syntax

```
#include "am_map.h"
AM_EXPORT boolean_t
am_map_entry_iter_is_entry_valid(am_map_entry_iter_t entry_iter);
```

Parameters

This function takes the following parameter:

`entry_iter` The specified entry iterator.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the standard `<types.h>` header file):

Note – The code used is dependent on the server operating system.

`!0` If the entry is valid.

`0` If the specified iterator is `NULL` or does not reference a valid entry.

am_map_entry_iter_next()

Advances the specified entry iterator to the next entry in the map specified when the iterator was first created.

Syntax

```
#include "am_map.h"
AM_EXPORT boolean_t
am_map_entry_iter_next(am_map_entry_iter_t entry_iter);
```

Parameters

This function takes the following parameter:

`entry_iter` The specified event iterator.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the standard `<types.h>` header file):

Note – The code used is dependent on the server operating system.

`!0`	If the entry is valid.
`0`	If the specified iterator is `NULL` or does not reference a valid entry after being updated.

am_map_erase()

Erases the entry, associated with the specified key, from the specified map object.

Syntax

```
#include "am_map.h"
AM_EXPORT am_status_t
am_map_erase(am_map_t map,
             const char *key);
```

Parameters

This function takes the following parameters:

`map`	The specified map object.
`key`	Pointer to the key of the entry to be erased.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the entry was successfully erased.
`AM_INVALID_ARGUMENT`	If either the map or key argument is `NULL`.
`AM_NOT_FOUND`	If the specified key is not in the map.

am_map_find()

Returns a value iterator that can sequence through all values associated with the specified key in the specified map object.

Syntax

```
#include "am_map.h"
AM_EXPORT am_status_t
am_map_find(am_map_t map,
            const char *key,
            am_map_value_iter_t *value_iter_ptr);
```

Parameters

This function takes the following parameters:

`map`	The specified map object.
`key`	Pointer to a key.
`value_iter_ptr`	Pointer specifying the location of the returned value iterator.

Note – If `value_iter_ptr` is not `NULL`, the location that it references will be set to `NULL` if an error is returned.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If no error was detected.
`AM_NO_MEMORY`	If unable to allocate memory for the key iterator.
`AM_INVALID_ARGUMENT`	If the `value_iter_ptr` argument is `NULL`.
`AM_NOT_FOUND`	If the specified key is not found in the map.

Memory Concerns

After using `value_iter_ptr`, the caller must call `am_map_value_iter_destroy()`.

am_map_find_first_value()

Returns the first value associated with the specified key in the specified map object.

Details

`am_map_find_first_value()` takes a key and returns the first value associated with that key.

Syntax

```
#include "am_map.h"
AM_EXPORT const char *
am_map_find_first_value(am_map_t map,
                        const char *key);
```

Parameters

This function takes the following parameters:

`map` The specified map object.

`key` Pointer to a key.

Returns

This function returns one of the following values:

`char *` Returns the first value associated with the specified key.

> **Note –** Caller must not modify or free the return value.

`NULL` If the specified key could not be found in the map or had no associated values.

am_map_for_each()

Returns a map iterator on a function pointer for the specified map object.

Details

`am_map_for_each()` will iterate over the list of key/value pairs and call each one. For every key in the map, a function can be invoked via the function pointer.

Syntax

```
#include "am_map.h"
AM_EXPORT am_status_t
am_map_for_each(am_map_t,
                am_status_t (*func)(const char *key,
                const char *value,
                void **args),
                void **args);
```

Parameters

This function takes the following parameters:

`am_map_t`	The specified map object.
`key`	Pointer to a key in an entry.
`args`	Pointer to application-defined parameters.

Returns

This function returns one of the following values:

Other codes	If the function returns any code other than `AM_SUCCESS`, the iteration will terminate and the same status code will be returned to the user.
`AM_INVALID_ARGUMENT`	If the parameters are invalid.

am_map_get_entries()

Returns an entry iterator object that can be used to enumerate all entries in the specified map.

Details

`am_map_get_entries()` returns a pointer to an entry iterator that can be used on the key/value pairs stored in the specified map object.

Syntax

```
#include "am_map.h"
AM_EXPORT am_status_t
am_map_get_entries(am_map_t map,
                   am_map_entry_iter_t *entry_iter_ptr);
```

Parameters

This function takes the following parameters:

`map`	The specified map object.
`entry_iter_ptr`	Pointer specifying the location of the entry iterator.

Note – If `entry_iter_ptr` is not `NULL`, the location it refers to will be set to `NULL` if an error is returned.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the <am_types.h> header file):

`AM_SUCCESS`	If no error is detected.
`AM_NO_MEMORY`	If unable to allocate memory for the entry iterator object.
`AM_INVALID_ARGUMENT`	If `entry_iter_ptr` is `NULL`.
`AM_NOT_FOUND`	If the specified map object contains no keys.

Memory Concerns

The pointer to the iterator must have only one iterator assigned. `am_map_entry_iter_destroy()` must be called when finished to destroy the iterator instance.

am_map_insert()

Inserts a new key/value pair into the specified map.

Details

The map does not retain any references to the provided key or value parameters. It makes copies of any strings it needs to store.

Syntax

```
#include "am_map.h"
AM_EXPORT am_status_t
am_map_insert(am_map_t map,
              const char *key,
              const char *value,
              int replace);
```

Parameters

This function takes the following parameters:

`map`	The specified map object.
`key`	Pointer to the key for the entry.

Note – If an entry with the same key already exists, the existing value is replaced by the new value.

`value`	Pointer to the [new] value to be associated with the key.
`replace`	If not zero, the specified value replaces all existing values. Otherwise, the specified value is added to the list of values already associated with the specified key.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the entry was successfully inserted into the map object.
`AM_NO_MEMORY`	If unable to allocate memory for the value and, if necessary, the key.
`AM_INVALID_ARGUMENT`	If either the map, key, or value argument is `NULL`.

am_map_size()

Returns the number of entries in the specified map object.

Syntax

```
#include "am_map.h"
AM_EXPORT size_t
am_map_size(const am_map_t map);
```

Parameters

This function takes the following parameter:

`map`	The specified map object.

Returns

This function returns a value based on `size_t` defined in the standard `<stddef.h>` header file. The value reflects the number of entries in the specified map object.

am_map_value_iter_destroy()

Destroys the specified value iterator.

Details

`am_map_value_iter_destroy()` destroys the `am_map_value_iter_t` passed to it. The caller must be sure that this function is not called multiple times on the same `am_map_value_iter_t`.

Syntax

```
#include "am_map.h"
AM_EXPORT void
am_map_value_iter_destroy(am_map_value_iter_t iter);
```

Parameters

This function takes the following parameter:

`iter` The specified value iterator.

Returns

This function does not return a value.

am_map_value_iter_get()

Returns the value currently referenced by the specified value iterator.

Syntax

```
#include "am_map.h"
AM_EXPORT const char *
am_map_value_iter_get(am_map_value_iter_t iter);
```

Parameters

This function takes the following parameter:

`iter` The specified value iterator.

Returns

This function returns one of the following values:

`char *` The value.

`NULL` If the specified iterator is `NULL` or does not reference a valid value.

am_map_value_iter_is_value_valid()

Determines if the specified value iterator references a valid value.

Syntax

```
#include "am_map.h"
AM_EXPORT boolean_t
am_map_value_iter_is_value_valid(am_map_value_iter_t iter);
```

Parameters

This function takes the following parameter:

`iter` The specified value iterator.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the standard `<types.h>` header file):

Note – The code used is dependent on the server operating system.

`!0` If the value is valid.

`0` If the specified iterator is `NULL` or does not reference a valid value.

am_map_value_iter_next()

Advances the specified value iterator to the next value associated with the key that was specified when the iterator was created.

Syntax

```
#include "am_map.h"
AM_EXPORT boolean_t
am_map_value_iter_next(am_map_value_iter_t iter);
```

Parameters

This function takes the following parameter:

`iter` The specified value iterator.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the standard `<types.h>` header file):

Note – The code used is dependent on the server operating system.

`!0` If successful.

`0` If the specified iterator is `NULL` or does not reference a valid value after being updated.

CHAPTER 7

Property Data Types and Functions

Sun™ Java System Access Manager contains public data types and functions you can use to associate properties between an external application and Access Manager. Reference summaries include a short description, syntax, parameters and returns. Prototypes for the types and functions are contained in the <am_properties.h> header file (located in the /*AccessManager-base*/`SUNWam/include` directory). This chapter contains the following sections:

- "Property Data Types" on page 131
- "Property Functions" on page 132

The Property API for C

The property API for C are used to manipulate configuration data read from a standard Java™ properties file. A properties file is a text file that contains a list of key/value pairs. More information on properties files can be found at `http://java.sun.com/j2se/1.4.2/docs/api/java/util/Properties.html#load(java.io.InputStream)`.

Property Data Types

The property data types defined in <am_properties.h> are:

- "`am_properties_t`" on page 131
- "`am_properties_iter_t`" on page 132

am_properties_t

Pointer to a properties object.

Details

`am_properties_t` provides a key to single value mapping. It provides the additional functionality of loading a configuration file and getting values of specific data types.

Syntax

```
#include "am_properties.h"
typedef struct am_properties *am_properties_t;
```

Members

`am_properties` is an opaque structure with no accessible members.

am_properties_iter_t

Pointer to the iterator for a properties object.

Syntax

```
#include "am_properties.h"
typedef struct am_properties_iter *am_properties_iter_t;
```

Members

`am_properties_iter` is an opaque structure with no accessible members.

Property Functions

The property functions defined in <am_properties.h> are:

am_properties_copy()

Duplicates a specified properties object.

Details

am_properties_copy() copies all the elements in the specified properties object, creates a duplicate instance, and assigns a pointer to it. The original object is not affected during the operation. The removal of any item in either structures does not affect the other.

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_copy(am_properties_t source_properties,
                   am_properties_t *properties_ptr);
```

Parameters

This function takes the following parameters:

source_properties	The specified properties object.
properties_ptr	Pointer to the location of the copy of the specified properties object.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the specified properties object was successfully copied.
AM_NO_MEMORY	If unable to allocate memory for the new properties object.
AM_INVALID_ARGUMENT	If the source_properties or properties_ptr argument is NULL.

Memory Concerns

After using the properties_ptr, call am_properties_destroy() to clean up the allocated memory.

am_properties_create()

Creates an empty properties object.

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_create(am_properties_t *properties_ptr);
```

Parameters

This function takes the following parameters:

`properties_ptr`	Pointer to the location of the new properties object.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If a properties object was successfully created.
`AM_NO_MEMORY`	If unable to allocate memory for the properties object.
`AM_INVALID_ARGUMENT`	If the `properties_ptr` argument is `NULL`.

Memory Concerns

After using the `properties_ptr`, call `am_properties_destroy()` to clean up the allocated memory.

am_properties_destroy()

Destroys the specified properties object.

Details

Be sure not to pass the same instance of `am_properties_t` to `am_properties_destroy()` more than once. After calling this function, it is advised to initialize `properties` to `NULL`.

Syntax

```
#include "am_properties.h"
AM_EXPORT void
am_properties_destroy(am_properties_t properties);
```

Parameters

This function takes the following parameter:

`properties`	Pointer to the specified properties object.

Returns

This function returns no values.

`am_properties_get()`

Retrieves the value associated with the specified key from the specified properties object.

Details

`am_properties_get()` checks for the presence of the specified key and returns its value, if present.

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_get(am_properties_t properties,
                  const char *key,
                  const char **value_ptr);
```

Parameters

This function takes the following parameters:

`properties`	Pointer to the specified properties object.
`key`	Pointer to the specified key in the specified properties object.
`value_ptr`	Pointer to a pointer to the location where the value associated with the specified `key` will be stored.

Returns

One of the following values as well as `value_ptr` containing an unparsed string with the address of the location of the value.

`AM_SUCCESS`	If no error is detected.
`AM_INVALID_ARGUMENT`	If the `properties`, `key`, or `value_ptr` argument is `NULL`.
`AM_NOT_FOUND`	If the specified key has no associated value and a default value is not provided.

`AM_INVALID_VALUE`	If the value associated with the specified key cannot be parsed as required by the particular accessor function.
`AM_NO_MEMORY`	If insufficient memory is available to look up the key.

Memory Concerns

Do not modify `value_ptr` or free the memory.

am_properties_get_boolean()

Retrieves boolean type values associated with the specified key from the specified properties object.

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_get_boolean(am_properties_t properties,
                          const char *key,
                          int *value_ptr);
```

Parameters

This function takes the following parameters:

`properties`	The specified properties object.
`key`	Pointer to the specified key in the specified properties object.
`value_ptr`	Pointer to the location where the boolean associated with the specified key will be stored.

Returns

One of the following values stored in `value_ptr`:

`!0`	If the value associated with the specified key is true, on, or yes.
`0`	If the value associated with the specified key is false, off, or no.

If the associated value does not match any of these recognized boolean values, `AM_INVALID_VALUE` will be returned.

am_properties_get_boolean_with_default()

Retrieves boolean type values from the specified properties object.

Details

am_properties_get_boolean_with_default() will return a defined default value if no other value is present, contrary to the behavior of am_properties_get_boolean().

Syntax

```
#include "am_properties.h"
am_properties_get_boolean_with_default(am_properties_t properties,
                                       const char *key,
                                       int default_value,
                                       int *value_ptr);
```

Parameters

This function takes the following parameters:

properties	The specified properties object.
key	Pointer to the specified key in the specified properties object.
default_value	Value to return if none is defined for the specified key.
value_ptr	Pointer to the location where the boolean associated with the specified key will be stored.

Returns

One of the following values stored in value_ptr:

!0	If the value associated with the specified key is true, on, or yes.
0	If the value associated with the specified key is false, off, or no.

If the associated value does not match any of the recognized boolean values, AM_INVALID_VALUE will be returned.

am_properties_get_entries()

Returns an iterator object that can be used to sequence through the entries in the specified properties object.

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_get_entries(am_properties_t properties,
                          am_properties_iter_t *properties_iter_ptr);
```

Parameters

This function takes the following parameters:

properties	The specified properties object.
properties_iter_ptr	Pointer to the location of the new properties iterator object.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If no error was detected.
AM_NO_MEMORY	If unable to allocate memory for the iterator object.
AM_INVALID_ARGUMENT	If properties_iter_ptr is NULL. If properties_iter_ptr is not NULL and an error is returned, the location that it refers to will be set to NULL.
AM_NOT_FOUND	If the specified properties object contains no entries.

am_properties_get_positive_number()

Retrieves a positive integer value associated with a specified key from the specified properties object.

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_get_unsigned(am_properties_t properties,
                           const char *key,
                           unsigned long default_value
                           unsigned long *value_ptr);
```

Parameters

This function takes the following parameters:

properties	Pointer to a properties object.
key	Pointer to the key in the properties object.
default_value	Value to return if none is defined for the specified key.
value_ptr	Pointer to the location where the returned integer will be stored.

Returns

This function returns the unsigned integer value associated with the specified key.

am_properties_get_signed()

Retrieves a signed integer value associated with the specified key from the specified properties object.

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_get_signed(am_properties_t properties,
                         const char *key,
                         long *value_ptr);
```

Parameters

This function takes the following parameters:

`properties`	The specified properties object.
`key`	Pointer to the specified key in the specified properties object.
`value_ptr`	Pointer to the location where the signed integer associated with the specified key will be stored.

Returns

This function returns a signed integer value associated with the specified key. If the associated value cannot be parsed as an integer or cannot be represented in the range `LONG_MIN` to `LONG_MAX`, `AM_INVALID_VALUE` will be returned.

am_properties_get_signed_with_default()

Retrieves a signed integer value associated with a specified key from the specified properties object.

Details

`am_properties_get_signed_with_default()` will return a defined default value if no other value is present, contrary to the behavior of `am_properties_get_signed()`.

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_get_signed_with_default(am_properties_t properties,
                                      const char *key,
                                      long default_value,
                                      long *value_ptr);
```

Parameters

This function takes the following parameters:

`properties`	The specified properties object.
`key`	Pointer to the specified key in the specified properties object.
`default_value`	Value to return if none is defined for the specified key.
`value_ptr`	Pointer to the location where the signed integer associated with the specified key will be stored.

Returns

This function returns a signed integer value associated with the specified key. If the associated value cannot be parsed as an integer or cannot be represented in the range `LONG_MIN` to `LONG_MAX`, `AM_INVALID_VALUE` will be returned.

am_properties_get_unsigned()

Retrieves an unsigned integer value associated with a specified key from the specified properties object.

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_get_unsigned(am_properties_t properties,
                           const char *key,
                           unsigned long *value_ptr);
```

Parameters

This function takes the following parameters:

`properties`	Pointer to a properties object.
`key`	Pointer to the key in the properties object.
`value_ptr`	Pointer to the location where the integer associated with the specified key will be stored.

Returns

This function returns the unsigned integer value associated with the specified key.

am_properties_get_unsigned_with_default()

Retrieves an unsigned integer value associated with a specified key from the specified properties object.

Details

am_properties_get_unsigned_with_default() will return a defined default value if no other value is present, contrary to the behavior of am_properties_get_unsigned().

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_get_unsigned_with_default(am_properties_t properties,
                                        const char *key,
                                        unsigned long default_value,
                                        unsigned long *value_ptr);
```

Parameters

This function takes the following parameters:

properties	The specified properties object.
key	Pointer to the specified key in the specified properties object.
default_value	Value to return if none is defined for the specified key.
value_ptr	Pointer to the location where the integer associated with the specified key will be stored.

Returns

This function returns the unsigned integer value associated with the specified key.

am_properties_get_with_default()

Retrieves the value (or the specified default) associated with the specified key from the specified properties object.

Details

am_properties_get_with_default() checks for the presence of the specified key and returns its value, if present. Contrary to am_properties_get(), if no value is present, it returns the specified default value.

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_get_with_default(am_properties_t properties,
                               const char *key,
                               const char *default_value,
                               const char **value_ptr);
```

Parameters

This function takes the following parameters:

`properties`	The specified properties object.
`key`	Pointer to the specified key in the specified properties object.
`default_value`	Pointer to the value to be returned in case of no associated value.
`value_ptr`	Pointer to a pointer to the location where the returned value will be stored.

Returns

One of the following values as well as `value_ptr` containing an unparsed string with the address of the location of the value.

`AM_SUCCESS`	If no error is detected.
`AM_INVALID_ARGUMENT`	If the properties, key, or `value_ptr` argument is NULL.
`AM_NOT_FOUND`	If the specified key has no associated value and a default value is not provided.
`AM_INVALID_VALUE`	If the value associated with the specified key is cannot be parsed as required by the particular accessor function.
`AM_NO_MEMORY`	If insufficient memory is available to look up the key.

Memory Concerns

Do not modify `value_ptr` or free the memory.

`am_properties_is_set()`

Determines whether the specified key of the specified properties object contains a value.

Details

`am_properties_is_set()` does not return the value.

Syntax

```
#include "am_properties.h"
AM_EXPORT boolean_t
am_properties_is_set(am_properties_t properties,
                     const char *key);
```

Parameters

This function takes the following parameters:

`properties`	The specified properties object.
`key`	Pointer to the name of a key.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the standard `<types.h>` header file):

`!0`	If the property has a value.
`0`	Otherwise

am_properties_iter_destroy()

Destroys the specified iterator object.

Syntax

```
#include "am_properties.h"
AM_EXPORT void
am_properties_iter_destroy(am_properties_iter_t properties_iter);
```

Parameters

This function takes the following parameter:

`properties_iter`	The specified iterator object. It may be `NULL`.

Returns

This function returns no value.

am_properties_iter_get_key()

Returns the key of the entry currently referenced by the specified iterator object.

Syntax

```
#include "am_properties.h"
AM_EXPORT const char *
am_properties_iter_get_key (am_properties_iter_t properties_iter);
```

Parameters

This function takes the following parameter:

`properties_iter`	The specified iterator object.

Returns

This function returns one of the following values:

`NULL`	If the specified iterator is `NULL` or does not reference a valid entry.
`char *`	The key.

am_properties_iter_get_value()

Returns the value of the key currently referenced by the specified iterator object.

Syntax

```
#include "am_properties.h"
AM_EXPORT const char *
am_properties_iter_get_value (am_properties_iter_t properties_iter);
```

Parameters

This function takes the following parameters:

`properties_iter`	The specified iterator object.

Returns

This function returns one of the following values:

`NULL`	If the specified iterator is `NULL` or does not reference a valid entry.
`char *`	Value associated with the key.

am_properties_load()

Loads information from the specified properties file into the specified properties object.

Details

The file is assumed to follow the syntax of a standard Java properties file.

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_load(am_properties_t properties,
                   const char *file_name);
```

Parameters

This function takes the following parameters:

`properties`	The specified properties object.
`file_name`	Pointer to a properties file.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If no error has occurred.
`AM_NOT_FOUND`	If the specified file does not exist.
`AM_NSPR_ERROR`	If there is a problem accessing the file.
`AM_INVALID_ARGUMENT`	If `properties` or `file_name` is `NULL` or `file_name` points to an empty string.
`AM_NO_MEMORY`	If unable to allocate memory to store the property information.

am_properties_set()

Sets a value for the specified key in the specified properties object.

Details

The specified value will replace any existing value.

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_set(am_properties_t properties,
                  const char *key,
                  const char *value);
```

Parameters

This function takes the following parameters:

`properties`	The specified properties object.
`key`	Pointer to the key being modified.
`value`	Pointer to the value to associate with the specified key.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If no error is detected.
`AM_INVALID_ARGUMENT`	If the properties, key, or value argument is `NULL`.
`AM_NO_MEMORY`	If unable to allocate memory to store the new key/value.

am_properties_store()

Retrieves key/value information from the specified properties object and stores it in the specified file.

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_store(am_properties_t properties,
                    const char *file_name);
```

Parameters

This function takes the following parameters:

`properties`	The specified properties object.
`file_name`	Pointer to the file in which the property information will be stored.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If no error is detected.
AM_NSPR_ERROR	If there is a problem writing to the file.
AM_INVALID_ARGUMENT	If properties or file_name is NULL or file_name points to an empty string.

CHAPTER 8

Web Agent Data Types and Functions

Sun™ Java System Access Manager contains public data types and functions intended for use by Sun Java System web agents. They can also be used to develop proprietary web agents. Reference summaries include a short description, syntax, parameters and returns. Prototypes for the types and functions are contained in the `<am_web.h>` header file (located in the `/`*AccessManager-base*`/SUNWam/include` directory). This chapter contains the following sections:

Web Agent API for C

The web agent application programming interface (API) for C are used by web agents to interact with Access Manager services such as the Authentication Service, the Session Service, the Policy Service, and the Logging Service. In order to use the data types and functions described herein, you should be familiar with web agents in general and how they work. The following books provide information for this purpose:

- *Sun Java System Access Manager 7.1 Technical Overview*
- *Sun Java System Access Manager Policy Agent 2.2 User's Guide*
- *Sun Java System Access Manager Policy Agent 2.2 Guide for Sun Java System Web Server 6.1*

Web Agent Data Types

The web agent data types defined in <am_web.h> are:

am_web_add_header_in_response_t

Defines a data type for the am_web_add_header_in_response_func_t function pointer.

Details

See also "am_web_add_header_in_response_func_t" on page 157.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_add_header_in_response_func_t func;
    void **args;
} am_web_add_header_in_response_t;
```

Members

The structure has the following components:

func   Pointer to am_web_add_header_in_response_func_t function.

args   Pointer to a pointer to agent defined parameters.

am_web_free_post_data_t

Defines a data type for the am_web_free_post_data_t function pointer.

Details

See also "am_web_free_post_data_func_t" on page 158.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_free_post_data_func_t func;
    void **args;
} am_web_free_post_data_t;
```

Members

am_web_free_post_data_t has the following components:

func	Pointer to am_web_free_post_data_func_t function.
args	Pointer to a pointer to agent defined parameters.

am_web_get_post_data_t

Defines a data type for the am_web_get_post_data_t function pointer.

Details

See also "am_web_get_post_data_func_t" on page 159.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_get_post_data_func_t func;
    void **args;
} am_web_get_post_data_t;
```

Members

am_web_get_post_data_t has the following components:

func	Pointer to am_web_get_post_data_func_t function.
args	Pointer to a pointer to agent defined parameters.

am_web_postcache_data_t

Data type for temporarily storing POST data sent by the web agent to the Access Manager Session Service.

Details

Policy agents use the POST method to communicate with the Session Service. For information, see "Session Validation" in *Sun Java System Access Manager 7.1 Technical Overview*.

Syntax

```
#include "am_web.h"
typedef struct am_web_postcache_data {
    char *value;
    char *url;
} am_web_postcache_data_t;
```

Members

`am_web_postcache_data_t` has the following components:

`value` Pointer to the string value of the POST data.

`url` Pointer to the destination URL of the POST.

am_web_render_result_t

Defines a data type for the `am_web_render_result_func_t` function pointer.

Details

See also "am_web_render_result_func_t" on page 160.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_render_result_func_t func;
    void **args;
} am_web_render_result_t;
```

Members

`am_web_render_result_t` has the following components:

`func`	Pointer to the `am_web_render_result_func_t` function.
`args`	Pointer to a pointer to agent defined parameters.

am_web_request_func_t

Defines an all-inclusive data type for the function pointers used by the web agent.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_get_post_data_t get_post_data;
    am_web_free_post_data_t free_post_data;
    am_web_set_user_t set_user;
    am_web_set_method_t set_method;
    am_web_set_header_in_request_t set_header_in_request;
    am_web_add_header_in_response_t add_header_in_response;
    am_web_render_result_t render_result;
} am_web_request_func_t;
```

Members

`am_web_request_func_t` has the following components:

`get_post_data`	See "`am_web_get_post_data_t`" on page 151.
`free_post_data`	See "`am_web_free_post_data_t`" on page 150.
`set_user`	See "`am_web_set_user_t`" on page 156.
`set_method`	See "`am_web_set_method_t`" on page 155.
`set_header_in_request`	See "`am_web_set_header_in_request_t`" on page 155.
`add_header_in_response`	See "`am_web_add_header_in_response_t`" on page 150.
`render_result`	See "`am_web_render_result_t`" on page 152.

am_web_request_params_t

Represents the parameters of an HTTP request passed to a web server from a client browser.

Details

This structure represents the parameters of the HTTP request and includes `am_web_req_method_t` which defines the action to be performed on the resource (GET, POST, DELETE, etc.).

Syntax

```
#include "am_web.h"
typedef struct {
    char *url;                      /* The full request URL */
    char *query;                    /* query string if any */
    am_web_req_method_t method;     /* request method */
    char *path_info;                /* path info if any */
    char *client_ip;                /* client IP if any */
    char *cookie_header_val;        /* the cookie header value if any */
    void *reserved;                 /* reserved - do not set this */
} am_web_request_params_t;
```

Members

am_web_request_params_t has the following components:

url
Pointer to the URL of the resource.

query
The query string appended to the request URL, if any. For example, if the URL is http://www.example.com?a=b&c=d, the value of this parameter would be a=b&c=d.

method
One of the following values of the am_web_req_method_t enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_WEB_REQUEST_UNKNOWN,
    AM_WEB_REQUEST_GET,
    AM_WEB_REQUEST_POST,
    AM_WEB_REQUEST_HEAD,
    AM_WEB_REQUEST_PUT,
    AM_WEB_REQUEST_DELETE,
    AM_WEB_REQUEST_TRACE,
    AM_WEB_REQUEST_OPTIONS
} am_web_req_method_t;
```

More information on these request methods can be found in http://www.faqs.org/rfcs/rfc2068.html.

path_info
The path information in the request URL, if any.

client_ip
Pointer to the IP address from which the request was sent.

cookie_header_val
Pointer to the cookie header.

reserved
Do not set this.

am_web_set_header_in_request_t

Defines a data type for the am_web_render_result_func_t function pointer.

Details

See also "am_web_render_result_func_t" on page 160.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_set_header_in_request_func_t func;
    void **args;
} am_web_set_header_in_request_t;
```

Members

am_web_set_header_in_request_t has the following components:

func	Pointer to am_web_set_header_in_request_func_t function.
args	Pointer to a pointer to agent defined parameters.

am_web_set_method_t

Defines a data type for the am_web_set_method_func_t function pointer.

Details

See also "am_web_set_method_func_t" on page 164.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_set_method_func_t func;
    void **args;
} am_web_set_method_t;
```

Members

am_web_set_method_t has the following components:

func	Pointer to the am_web_set_method_func_t function.
args	Pointer to a pointer to agent defined parameters.

am_web_set_user_t

Defines a data type for the am_web_set_user_func_t function pointer.

Details

See also "am_web_set_user_func_t" on page 165.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_set_user_func_t func;
    void **args;
} am_web_set_user_t;
```

Members

am_web_set_user_t has the following components:

func	Pointer to am_web_set_user_func_t function.
args	Pointer to a pointer to agent defined parameters.

post_urls_t

A session information object defining the URLs used by the web agent to communicate with Access Manager.

Syntax

```
#include "am_web.h"
typedef struct post_urls {
    char *dummy_url;
    char *action_url;
    char *post_time_key;
} post_urls_t;
```

Members

post_urls_t has the following components:

dummy_url	Pointer to a dummy URL to redirect for POST data preservation.

Note – *POST data preservation* is supported only on Policy Agent 2.2 for Sun Java System Web Server 6.1. The feature allows for submitted POST data to be preserved. See "Preserving POST Data on Sun Java System Web Server 6.1 Only" in *Sun Java System Access Manager Policy Agent 2.2 Guide for Sun Java System Web Server 6.1* for more information.

`action_url`	Pointer to destination URL for a POST request.
`post_time_key`	Pointer to a unique key used to tag a POST data entry.

Web Agent Function Pointers

The web agent function pointers must be written before calling the `am_web_process_request()` function to process a request. The function pointers defined in <am_web.h> are:

- "`am_web_add_header_in_response_func_t`" on page 157
- "`am_web_free_post_data_func_t`" on page 158
- "`am_web_get_cookie_sync_func_t`" on page 159
- "`am_web_get_post_data_func_t`" on page 159
- "`am_web_render_result_func_t`" on page 160
- "`am_web_result_set_header_func_t`" on page 161
- "`am_web_result_set_header_attr_in_request_func_t`" on page 162
- "`am_web_result_set_header_attr_in_response_func_t`" on page 163
- "`am_web_set_header_in_request_func_t`" on page 164
- "`am_web_set_method_func_t`" on page 164
- "`am_web_set_user_func_t`" on page 165

am_web_add_header_in_response_func_t

Adds (or sets) an HTTP header in a response.

Details

If a header of the same name already exists, it should be replaced with this header.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_add_header_in_response_func_t)
                              (void **args,
                               const char *name,
                               const char *val);
```

Parameters

This function takes the following parameter:

`args`	Pointer to a pointer to agent defined parameters.
`name`	Pointer to the name of the header.
`val`	Pointer to the value of the header.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the header was successfully added.
Appropriate `am_status_t` code	Otherwise.

am_web_free_post_data_func_t

Frees the data retrieved by `am_web_get_post_data_func_t`.

Details

The POST data can be `NULL` if it is not needed.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_free_post_data_func_t)
                               (void **args,
                                char *data);
```

Parameters

This function takes the following parameter:

`args`	Pointer to a pointer to agent defined parameters.
`data`	Pointer to the data.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the data was successfully freed.

Warning	If not successfully freed, the status will be logged as a warning but ignored.

am_web_get_cookie_sync_func_t

Synchronizes two cookies.

Details

Currently, this is a dummy function. Do not use.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_get_cookie_sync_func_t)(
                                const char *cookieName,
                                char **dproCookie,
                                void **args);
```

Parameters

This function takes the following parameter:

`cookieName`	Pointer to the cookie with which the Access Manager cookie will be synchronized.
`dproCookie`	Pointer to a pointer to the Access Manager cookie.
`args`	Pointer to a pointer to agent defined parameters.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the data was successfully freed.
Warning	If not successfully freed, the status will be logged as a warning but ignored.

am_web_get_post_data_func_t

Retrieves post data.

Details

The returned POST data must be `NULL` terminated and will be freed by calling `am_web_free_post_data_func_t`.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_get_post_data_func_t)
                              (void **args,
                               char **data);
```

Parameters

This function takes the following parameter:

`args`	Pointer to a pointer to agent defined parameters.
`data`	Pointer to a pointer to the data.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the data was successfully retrieved.
HTTP internal error	Otherwise

am_web_render_result_func_t

Renders an HTML page based on the result of a web agent's enforcement.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_render_result_func_t)
                              (void **args,
                               am_web_result_t http_result,
                               char *data);
```

Parameters

This function takes the following parameter:

`args`	Pointer to a pointer to agent defined parameters.
`http_result`	One of the following values of the `am_web_result_t` enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_WEB_RESULT_OK,               /* access check was OK */
    AM_WEB_RESULT_OK_DONE,          /* OK and handled (for ex. notification) */
```

```
    AM_WEB_RESULT_FORBIDDEN,        /* access forbidden */
    AM_WEB_RESULT_REDIRECT,         /* redirected */
    AM_WEB_RESULT_ERROR             /* internal error */
} am_web_result_t;
```

For AM_WEB_RESULT_OK_DONE, the web agent should return an HTTP status code 200 OK and the body of the HTTP response should be set to the string in the data parameter. For AM_WEB_RESULT_REDIRECT, the web agent should return an HTTP status code 302 and the Location header should be set to the redirect URL in the data argument. More information on these request methods can be found in http://www.faqs.org/rfcs/rfc2068.html.

data — Pointer to a string defining user data, a URL, or other data.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS — If page was successfully rendered.

Warning — If not successfully freed, the status will be logged as a warning but ignored.

am_web_result_set_header_func_t

Sets LDAP attributes in an HTTP header.

Details

This function will be called when com.sun.am.policy.agents.config.profile.attribute.fetch.mode in AMAgent.properties is set to HTTP_HEADER. This property specifies if additional user profile attributes should be introduced into the request. Possible values are:

- NONE
- HTTP_HEADER
- HTTP_COOKIE

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_result_set_header_func_t)(
                                 const char *key,
                                 const char *attrValues,
                                 void **args);
```

Parameters

This function takes the following parameter:

`key`	Pointer to the key to which the value will be set.
`attrValues`	Pointer to a string representing the values to be set.
`args`	Pointer to a pointer to agent defined parameters.

Returns

This function returns one of the values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file) including:

`AM_SUCCESS`	If the header was successfully set.

am_web_result_set_header_attr_in_request_func_t

Sets LDAP attributes defined in the request's HTTP cookie header.

Details

This function will be called when `com.sun.am.policy.agents.config.profile.attribute.fetch.mode` in `AMAgent.properties` is set to `HTTP_COOKIE`. This property specifies if additional user profile attributes should be introduced into the request. Possible values are:

- `NONE`
- `HTTP_HEADER`
- `HTTP_COOKIE`

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_result_set_header_attr_in_request_func_t)(
                                const char *cookieValues,
                                void **args);
```

Parameters

This function takes the following parameter:

`cookieValues`	Pointer to string representing the values in the cookie.
`args`	Pointer to a pointer to agent defined parameters.

Returns

This function returns one of the values of the am_status_t enumeration (defined in the <am_types.h> header file) including:

AM_SUCCESS If the header was successfully set.

am_web_result_set_header_attr_in_response_func_t

Sets LDAP attributes defined in the response's HTTP cookie header.

Details

This function will be called when com.sun.am.policy.agents.config.profile.attribute.fetch.mode in AMAgent.properties is set to HTTP_COOKIE. This property specifies if additional user profile attributes should be introduced into the request. Possible values are:

- NONE
- HTTP_HEADER
- HTTP_COOKIE

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_result_set_header_attr_in_response_func_t)(
                     const char *cookieValues,
                     void **args);
```

Parameters

This function takes the following parameter:

cookieValues	Pointer to string representing the values in the cookie.
args	Pointer to a pointer to agent defined parameters.

Returns

This function returns one of the values of the am_status_t enumeration (defined in the <am_types.h> header file) including:

AM_SUCCESS If the header was successfully set.

am_web_set_header_in_request_func_t

Sets an HTTP header in a request.

Details

If a header of the same name already exists it should be replaced with this header.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_set_header_in_request_func_t)(
                     void **args,
                     const char *name,
                     const char *val);
```

Parameters

This function takes the following parameter:

`args`	Pointer to a pointer to agent defined parameters.
`name`	Pointer to the name of the header.
`val`	Pointer to the value of the header.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the <am_types.h> header file):

`AM_SUCCESS`	If header was successfully added.
Warning	If not successfully freed, the status will be logged as a warning but ignored.

am_web_set_method_func_t

Sets the request method to be used by a web agent during cross domain single sign-on (CDSSO).

Details

In cases of CDSSO actions between Access Manager and web agents, the POST request method is used by the web agent. `am_web_set_method_func_t` is required to change the request method to POST, if necessary, from the method defined in the original HTTP request. See "Cross-Domain Single Sign-On Session" in *Sun Java System Access Manager 7.1 Technical Overview* for additional information.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_set_method_func_t)(
                                void **args,
                                am_web_req_method_t method);
```

Parameters

This function takes the following parameter:

`args`	Pointer to a pointer to agent defined parameters.
`method`	One of the following values of the `am_web_req_method_t` enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_WEB_REQUEST_UNKNOWN,
    AM_WEB_REQUEST_GET,
    AM_WEB_REQUEST_POST,
    AM_WEB_REQUEST_HEAD,
    AM_WEB_REQUEST_PUT,
    AM_WEB_REQUEST_DELETE,
    AM_WEB_REQUEST_TRACE,
    AM_WEB_REQUEST_OPTIONS
} am_web_req_method_t;
```

More information on these request methods can be found in `http://www.faqs.org/rfcs/rfc2068.html`.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the method was successfully set.
HTTP forbidden result	Otherwise.

`am_web_set_user_func_t`

Sets the user.

Details

The implementation code sets the user.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_set_user_func_t)(
                     void **args,
                     const char *user);
```

Parameters

This function takes the following parameter:

`args`	Pointer to a pointer to agent defined parameters.
`user`	Pointer to the user login.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the <am_types.h> header file):

`AM_SUCCESS`	If user was successfully set.
HTTP forbidden result	Otherwise.

Web Agent Functions

The web agent functions defined in <am_web.h> are:

am_web_build_advice_response()

Builds an advice response.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
```

```
am_web_build_advice_response(const am_policy_result_t *policy_result,
                             const char *redirect_url,
                             char **advice_response);
```

Parameters

This function takes the following parameter:

`policy_result`	Pointer to an `am_policy_result_t` data type.

Note – See "am_policy_result_t" on page 47 for information.

`redirect_url`	Pointer to a redirect URL.
`advice_response`	Pointer to a pointer to the location of the advice response.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the <am_types.h> header file):

`AM_SUCCESS`	If the query parameter was found in the URL.
`AM_*`	If any other error occurred.

am_web_check_cookie_in_post()

Retrieves a user's `SSOToken`.

Details

In cases of cross domain single sign-on, Access Manager sends out a user's `SSOToken` using POST. This method uses POST to retrieve the `SSOToken` and set it in the foreign domain.

Syntax

```
#include "am_web.h"
M_WEB_EXPORT am_status_t
am_web_check_cookie_in_post(void ** args,
                            char ** dpro_cookie,
                            char ** request_url,
                            char **orig_req,
                            char *method,
                            char *response,
                            boolean_t responseIsCookie,
               am_status_t (*set_cookie)(const char *, void **),
```

```
                     void (*set_method)(void **, char *)
);
```

Parameters

This function takes the following parameter:

`args`	Pointer to a pointer to agent defined parameters.
`dpro_cookie`	Pointer to a pointer to the Access Manager cookie.
`request_url`	Pointer to a pointer to the CDSSO URL.
`orig_req`	Pointer to a pointer to the original request method.
`method`	Pointer to the changed method name.
`response`	Pointer to the response which will hold the POST data.
`responseIsCookie`	Returns one of the following values of the `boolean_t` enumeration (defined in the `<am_types.h>` header file): `B_TRUE` If using Liberty Alliance Project specifications. `B_FALSE` If Access Manager POST data.
`set_cookie`	Function pointer used to set the cookie in the foreign domain.
`set_method`	Function pointer used to reset the original method in the request.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the query parameter was found in the URL.
`AM_*`	If any other error occurred.

`am_web_check_cookie_in_query()`

Retrieves the cookie from a query string.

Details

In older versions of Access Manager, when performing CDSSO, the cookie was part of the query string.

Syntax

```
AM_WEB_EXPORT am_status_t
am_web_check_cookie_in_query(void **args,
                             char **dpro_cookie,
                             const char *query,
                             char **request_url,
                             char ** orig_req,
                             char *method,
                am_status_t (*set_cookie)(const char *, void **),
                       void (*set_method)(void **, char *)
);
```

Parameters

This function takes the following parameter:

`args`	Pointer to a pointer to agent defined parameters.
`dpro_cookie`	Pointer to a pointer to the Access Manager cookie.
`query`	Pointer to the query.
`request_url`	Pointer to a pointer to the CDSSO URL.
`orig_req`	Pointer to a pointer to the original request method.
`method`	Pointer to a pointer to the changed method name.
`set_cookie`	Function pointer used to set the cookie in the foreign domain.
`set_method`	Function pointer used to reset the original method in the request.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the query parameter was found in the URL.
`AM_*`	If any other error occurred.

`am_web_clean_post_urls()`

Cleans up a `post_urls_t` data type.

Details

See “`post_urls_t`” on page 156 for more information.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_clean_post_urls(post_urls_t *posturl_struct);
```

Parameters

This function takes the following parameter:

`posturl_struct`	Pointer to `post_urls_t` data type.

Returns

This function returns no values.

`am_web_cleanup()`

Cleans up any memory called by the `am_web_*` functions.

Details

This should be called before a web agent exits.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_cleanup();
```

Parameters

This function does not take any parameters.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the <am_types.h> header file):

`AM_SUCCESS`	If the cleanup was successful.
`AM_*`	If any error occurs, the type of error indicated by the status value.

am_web_create_post_page()

Creates an HTML form that submits the POST data with invisible name/value pairs.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT char *
am_web_create_post_page(const char *key,
                        const char *postdata,
                        const char *actionurl);
```

Parameters

This function takes the following parameters:

`key`	Pointer to the unique key identifying a POST data entry. It is used to remove post data once the page is re-posted.
`postdata`	Pointer to a browser encoded string representing the POST data entry.
`actionurl`	Pointer to the POST destination URL.

Returns

This function returns `char *` as the POST form to be submitted.

am_web_create_post_preserve_urls()

Constructs a `post_urls_t` data type during preservation of POST data.

Details

A `post_urls_t` data type contains a dummy POST URL, an action URL and a unique key. The dummy URL is filtered by the Server Application Function (SAF) to identify POST preservation redirects from general redirects. All three of these variables are required for POST preservation.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT post_urls_t *
am_web_create_post_preserve_urls(const char *request_url);
```

Parameters

This function takes the following parameter:

`request_url` Pointer to the request URL for POST in the HTTP request.

Returns

This function returns a `post_urls_t` data type. See “post_urls_t” on page 156 for information.

am_web_do_cookie_domain_set()

Sets the Access Manager cookie (called `iPlanetDirectoryPro`) for each domain configured in the `com.sun.am.policy.agents.config.cookie.domain.list` property in `AMAgent.properties`.

Details

`am_web_do_cookie_domain_set()` builds the set-cookie header for each domain, and calls the callback function declared in `setFunc` to set the cookie. In CDSSO, the callback function is called by `am_web_check_cookie_in_query()` and `am_web_check_cookie_in_post()` to set the cookie in the response.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_do_cookie_domain_set(am_status_t (*setFunc)(
                                        const char *,
                                        void **),
                                        void **args,
                                        const char *cookie);
```

Parameters

This function takes the following parameters:

`setFunc` Function pointer with which the user can define their own function for setting the cookie in the foreign domain. The implementation defines the parameters.

`args` Pointer to a pointer to agent defined parameters.

`cookie` Pointer to the cookie.

Returns

This function returns a value of the `am_status_t` enumeration (defined in the `<am_types.h>` header file).

am_web_do_cookies_reset()

Resets the cookies in a response before redirecting it for authentication to the Access Manager login page.

Details

This function resets the cookies specified in `AMAgent.properties` and invokes the set action that the agent passes in for each of them. It is enabled by setting the following properties:

- `com.sun.am.policy.agents.config.cookie.reset.enable`: This property must be set to `true` if the web agent needs to reset cookies in the response before redirecting them to Access Manager for Authentication. By default it is set to `false`.
- `com.sun.am.policy.agents.config.cookie.reset.list`: This property (used only if `com.sun.am.policy.agents.config.cookie.reset.enable` is enabled) contains a comma-separated list of cookies that need to be included in the response redirected to Access Manager.

See the `AMAgent.properties` file for more information.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_do_cookies_reset(am_status_t (*setFunc)(
                                     const char *,
                                     void **),
                                     void **args);
```

Parameters

This function takes the following parameters:

`setFunc`	Function pointer with which the user can define their own function for setting the cookie in the foreign domain. The implementation defines the parameters.
`args`	Pointer to a pointer to agent defined parameters.

Returns

This function returns a value of the `am_status_t` enumeration (defined in the `<am_types.h>` header file).

am_web_free_memory()

Frees memory previously allocated by a `am_web_*` routine.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_free_memory(void *memory);
```

Parameters

This function takes the following parameter:

`memory`	Pointer to the memory.

Returns

This function returns no value.

am_web_get_agent_server_host()

Retrieves the name of the server host for the agent.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT const char *
am_web_get_agent_server_host();
```

Parameters

This function takes no parameters.

Returns

This function returns then name of the server host.

am_web_get_agent_server_port()

Retrieves the port used by the agent on the server host.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT int
am_web_get_agent_server_port();
```

Parameters

This function takes no parameters.

Returns

This function returns a port number.

am_web_get_authType()

Determines if the auth-type value DSAME should be replaced by Basic in the Sun Java System Access Manager Policy Agent 2.2 for Microsoft IIS 6.0.

Details

The Sun Java System Access Manager Policy Agent 2.2 for Microsoft IIS 6.0 is defined in <am_web.h> as having auth-type value equal to DSAME.

```
#define     AM_WEB_AUTH_TYPE_VALUE     "DSAME"
```

DSAME is a hard-coded value representing all Access Manager authentication types other than Basic.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT const char *
am_web_get_authType();
```

Parameters

This function takes no parameters.

Returns

This function returns either DSAME or Basic.

am_web_get_cookie_name()

Retrieves the name of the Access Manager cookie.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT const char *
am_web_get_cookie_name();
```

Parameters

This function takes no parameters.

Returns

This function returns the name of the Access Manager cookie.

am_web_get_notification_url()

Retrieves the URL of the Access Manager Notification Service.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT const char *
am_web_get_notification_url();
```

Parameters

This function does not take any parameters.

Returns

This function returns the URL of the Access Manager Notification Service.

am_web_get_parameter_value()

Gets the value of the specified parameter from the specified request URL.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_get_parameter_value(const char *inpQuery,
                           const char *param_name,
                           char **param_value);
```

Parameters

This function takes the following parameters:

`inpQuery`	Pointer to the request URL that holds the parameter.
`param_name`	Pointer to the name of the parameter.
`param_value`	Pointer to a pointer to be filled with the value of the `param_name` parameter, if found.

Returns

This function also returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the query parameter was found in the URL.
`AM_*`	If any other error occurred.

Memory Concerns

The returned parameter value should be freed by the caller using `am_web_free()`.

am_web_get_request_url()

Parses the host request header field for a server host name, port, protocol, query parameter, and URI to return the requested URL to the web agent.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_get_request_url(const char *host_hdr,
                       const char *protocol,
                       const char *hostname,
                       size_t port,
                       const char *uri,
                       const char *query,
                       char **req_url);
```

Parameters

This function takes the following parameters:

`host_hdr`	Pointer to the host header string of the HTTP request as passed from the browser.
`protocol`	Pointer to the protocol used by the web container of the resource being requested.

`hostname`	Pointer to the name of the host on which the resource being requested.
`port`	Value based on the `size_t` defined in the standard `<stddef.h>` header file that reflects the port number of the resource being requested.
`uri`	Pointer to the URI of the HTTP request.

Note – Most URLs have this basic form: *protocol://server:port/request-URI*. The *request-URI* portion of the URL is used by the web server to identify the document.

`query`	The query string appended to the request URL, if any. For example, if the URL is `http://www.example.com?a=b&c=d`, the value of this parameter would be `a=b&c=d`.
`req_url`	Pointer to a pointer to the OUT parameter to be populated with the value of the URL string to be used by the agent.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the query parameter was found in the URL.
`AM_*`	If any other error occurred.

am_web_get_url_to_redirect()

Returns a string representing a login URL or access denied URL to which the web agent should redirect.

Caution – `am_web_get_redirect_url()` has been deprecated and must not be used. It is supported for backward compatibility only.

Details

`am_web_get_url_to_redirect()` may redirect the user to the login URL or the access denied URL. The URL is appropriate to the provided status code and advice map returned by the Policy SDK. If redirecting to the login URL, the URL will include existing information specified in the URL from the configuration file (for example, the organization name) as well as the specified `goto` parameter value which will be used by Access Manager after the user has successfully authenticated. The last parameter reserved must be passed with `NULL`.

Note – If the URL returned is not `NULL`, the caller of this function must call `am_web_free_memory(void *)` to free the pointer.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_get_url_to_redirect(am_status_t status,
                           const am_map_t advice_map,
                           const char *goto_url,
                           const char* method,
                           void *reserved,
                           char ** redirect_url);
```

Parameters

This function takes the following parameters:

`status` The status from `am_web_is_access_allowed()`.

Note – See "`am_web_is_access_allowed()`" on page 183.

`advice_map` Any advice map from policy evaluation results.

`goto_url` Pointer to the original URL which the user attempted to access.

`method` Pointer to the original HTTP method: GET or POST.

`reserved` This parameter is not currently used.

`redirect_url` Pointer to a pointer containing the resulting Access Manager redirect URL.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS` If successful.

`AM_*` Otherwise.

am_web_get_token_from_assertion()

Returns the single sign-on token from the specified Security Assertion Markup Language (SAML) assertion.

Details

`am_web_get_token_from_assertion()` is used to retrieve the cookie sent by Access Manager in a SAML assertion.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_get_token_from_assertion(char *assertion,
                                char **token);
```

Parameters

This function takes the following parameters:

`assertion`	Pointer to the SAML assertion as an XML string.
`token`	Pointer to a pointer containing the single sign-on token identifier.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If successful.
`AM_*`	Otherwise.

Memory Concerns

The returned identifier should be freed using `am_web_free()`.

am_web_handle_notification()

Handles notification data received by a web agent.

Details

`am_web_handle_notification()` generates logging messages for the event and any error that may occur during the processing of the notification.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_handle_notification(const char *data,
                           size_t data_length);
```

Parameters

This function takes the following parameters:

`data` Pointer to the notification message as an XML string.

`data_length` Value based on the `size_t` defined in the standard `<stddef.h>` header file that reflects the length of the notification message.

Returns

This function returns no value.

am_web_http_decode()

URL decodes the specified URL encoded string.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT char *
am_web_http_decode(const char *string,
                   size_t len);
```

Parameters

This function takes the following parameters:

`string` Pointer to the URL encoded string.

`len` Value based on the `size_t` defined in the standard `<stddef.h>` header file that reflects the length of the string.

Returns

This function returns the URL decoded value of the URL encoded string, or `NULL` if an error occurred.

Memory Concerns

The returned value should be freed by calling `am_web_free()`.

am_web_init()

Initializes the web agent.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_init(const char *config_file);
```

Parameters

This function takes the following parameter:

`config_file`	Pointer to the agent configuration file as in `/etc/opt/AMAgent.properties`.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the <am_types.h> header file):

`AM_SUCCESS`	If the call was successful.
`AM_*`	If any error occurs, the type of error indicated by the status value.

am_web_is_access_allowed()

Evaluates the access control policies for a specified web resource and action against those for a specified user.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_is_access_allowed(const char *sso_token,
                         const char *url,
                         const char *path_info,
                         const char *action_name,
                         const char *client_ip,
                         const am_map_t env_parameter_map,
                         am_policy_result_t *result);
```

Parameters

This function takes the following parameters:

`sso_token`	Pointer to the session token from the Access Manager cookie. This parameter may be `NULL` if there is no cookie present.
`url`	Pointer to the web resource URL. This parameter may not be `NULL`.
`path_info`	Pointer to the path information in the web resource URL, if any.

action_name	Pointer to the action (GET, POST, etc.) being performed on the specified resource URL. This parameter may not be NULL.
client_ip	Pointer to the IP address of the client attempting to access the specified resource URL. If client IP validation is turned on, this parameter may not be NULL.
env_parameter_map	A map object containing additional information about the user attempting to access the specified resource URL. This parameter may not be NULL.
advices_map_ptr	An output parameter where the am_map_t can be stored if the policy evaluation produces any advice information. This parameter may not be NULL. See "am_map_t" on page 113 for more information.
result	Pointer to a policy result object.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the evaluation was performed successfully and access is allowed.
AM_NO_MEMORY	If the evaluation was not successfully completed due to insufficient memory being available.
AM_INVALID_ARGUMENT	If any of the url, action_name, env_parameter_map, or advices_map_ptr parameters is NULL or if client IP validation is enabled and the client_ip parameter is NULL.
AM_INVALID_SESSION	If the specified session token does not refer to a currently valid session
AM_ACCESS_DENIED	If the policy information indicates that the user does not have permission to access the specified resource or any error is detected other than the ones listed above.

am_web_is_cdsso_enabled()

Returns a boolean specifying whether cross domain single sign-on is enabled in the agent's configuration file.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_cdsso_enabled();
```

Parameters

This function takes no parameters.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the <am_types.h> header file):

`B_TRUE`	If cross domain single sign-on is enabled.
`B_FALSE`	Otherwise.

`am_web_is_cookie_present()`

Detects whether a cookie is present.

Details

This function will most probably be invoked in a loop. A cookie name and value is passed and the implementation checks whether the cookie is already listed. If not, the new cookie name and value are appended. If present, the value of the cookie name is updated.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT int
am_web_is_cookie_present(const char *cookie,
                         const char *value,
                         char **new_cookie);
```

Parameters

This function takes the following parameters:

`cookie`	Pointer to a cookie.
`value`	Pointer to a value.
`new_cookie`	Pointer to a pointer to the location of the new cookie.

Returns

This function returns one of the following integers as defined in <am_web.h>:

```
#define AM_WEB_COOKIE_EXIST            2
#define AM_WEB_COOKIE_MODIFIED         1
```

```
#define AM_WEB_COOKIE_ABSENT            0
#define AM_WEB_COOKIE_ERROR            -1
```

am_web_is_debug_on()

Returns a boolean specifying whether debug is enabled.

Details

`am_web_is_debug_on()` specifies whether the log level is set to anything greater than 0.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_debug_on();
```

Parameters

This function takes no parameters.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the `<am_types.h>` header file):

`B_TRUE` If the log level is set to anything greater than 0.

`B_FALSE` Otherwise.

am_web_is_in_not_enforced_ip_list()

Returns a boolean specifying whether the given IP address is defined as one where no authentication or authorization is required.

Details

IP addresses that are not enforced are defined in the `com.sun.am.policy.agents.config.notenforced_client_ip_list` property in `AMAgent.properties`. If the IP address from where the request was issued is not enforced, the request goes through without authentication or authorization.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_in_not_enforced_ip_list(const char *ip);
```

Parameters

This function takes the following parameter:

`ip` Pointer to the IP address.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the `<am_types.h>` header file):

`B_TRUE` If the IP is in the not enforced IP address list.

`B_FALSE` Otherwise.

am_web_is_in_not_enforced_list()

Returns a boolean specifying whether the URL being accessed by the user is in the not enforced list.

Details

URLs that are not enforced are defined in the `com.sun.am.policy.agents.config.notenforced_list` property in `AMAgent.properties`. If the URL is not enforced, the request goes through without authentication or authorization.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_in_not_enforced_list(const char *url,
                               const char *path_info);
```

Parameters

This function takes the following parameters:

`url` Pointer to the URL being accessed.

`path_info` Pointer to the path information in the URL being accessed, if any.

Returns

This function returns one of the following values of the boolean_t enumeration (defined in the <am_types.h> header file):

B_TRUE If the URL is in the not enforced list.

B_FALSE Otherwise.

am_web_is_logut_url()

Returns a boolean specifying whether the specified URL is a logout URL.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_logut_url(const char *url);
```

Parameters

This function takes the following parameter:

url Pointer to a URL.

Returns

This function returns one of the following values of the boolean_t enumeration (defined in the <am_types.h> header file):

B_TRUE If the specified URL is a logout URL.

B_FALSE Otherwise.

am_web_is_max_debug_on()

Returns a boolean specifying whether the log level is set to 5.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_max_debug_on();
```

Parameters

This function takes no parameters.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the `<am_types.h>` header file):

`B_TRUE` If the log level is set to 5.

`B_FALSE` Otherwise.

`am_web_is_notification()`

Returns a boolean specifying whether the given URL is the Notification Service URL for the web agent as configured in `AMAgent.properties`.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_notification(const char *request_url);
```

Parameters

This function takes the following parameter:

`request_url` Pointer to the Notification Service URL

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the `<am_types.h>` header file):

`B_TRUE` If the URL is the Notification Service URL of the agent as set in `AMAgent.Properties`.

`B_FALSE` Otherwise.

`am_web_is_postpreserve_enabled()`

Returns a boolean specifying whether POST data preservation is enabled for clients in `AMAgent.Properties`.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_postpreserve_enabled();
```

Parameters

This function takes no parameters

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the `<am_types.h>` header file):

`B_TRUE`	If POST data preservation is on.
`B_FALSE`	If POST data preservation is off.

am_web_is_proxy_override_host_port_set()

Determines if the `com.sun.am.policy.agents.config.override_host` and the `com.sun.am.policy.agents.config.override_port` properties are set for the proxy agent.

Details

These properties are defined in `AMAgent.properties`.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t am_web_is_proxy_override_host_port_set();
```

Parameters

This function takes no parameters.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the `<am_types.h>` header file):

`B_TRUE`	If set.
`B_FALSE`	Otherwise.

am_web_is_valid_fqdn_url()

Returns a boolean specifying whether the requested URL is a valid fully qualified domain name (FQDN) resource as configured in `AMAgent.properties`. For example, `myhost.mydomain.com`.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_valid_fqdn_url(const char *url);
```

Parameters

This function takes no parameters.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the <am_types.h> header file):

`B_TRUE` If the URL is using a fully qualified domain name.

`B_FALSE` Otherwise.

am_web_log_always()

Log the given message regardless of the log level set in `AMAgent.properties`.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_log_always(const char *fmt, ...);
```

Parameters

This function takes the following parameter:

`fmt` Pointer to a formatted string message as in `printf`.

Returns

This function returns no values.

am_web_log_auth()

Log the given access allowed or denied message to the Access Manager logs.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_log_auth(am_web_access_t access_type,
                const char *fmt, ...);
```

Parameters

This function takes the following parameters:

`access_type` One of the following values of the `am_web_access_t` enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_ACCESS_DENY = 0,
    AM_ACCESS_ALLOW
} am_web_access_t;
```

`fmt` Pointer to a formatted string message as in `printf`.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the `<am_types.h>` header file):

`B_TRUE` If the call was successful.

`B_FALSE` Otherwise.

am_web_log_debug()

Log the given message at the debug level.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_log_debug(const char *fmt, ...);
```

Parameters

This function takes the following parameter:

`fmt` Pointer to a formatted string message as in `printf`.

Returns

This function returns no values.

am_web_log_error()

Log the given message at the debug log level.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_log_error(const char *fmt, ...);
```

Parameters

This function takes the following parameter:

`fmt` Pointer to a formatted string message as in `printf`.

Returns

This function returns no values.

am_web_log_info()

Log the given message at the info log level.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_log_info(const char *fmt, ...);
```

Parameters

This function takes the following parameter:

`fmt` Pointer to a formatted string message as in `printf`.

Returns

This function returns no values.

am_web_log_max_debug()

Log the given message at maximum debug level.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_log_max_debug(const char *fmt, ...);
```

Parameters

This function takes the following parameters:

`fmt` Pointer to a formatted string message as in `printf`.

Returns

This function returns no values.

am_web_log_warning()

Log the given message at the warning log level.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_log_warning(const char *fmt, ...);
```

Parameters

This function takes the following parameters:

`fmt` Pointer to a formatted string message as in `printf`.

Returns

This function returns no values.

am_web_logout_cookies_reset()

Resets cookie configured for reset on user logout.

Details

The reset function passed in is called for each cookie configured for reset. If the function failed for any cookie, the last failed status is returned.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_logout_cookies_reset(am_status_t (*setFunc)(
                                     const char *,
                                     void **),
                                     void **args);
```

Parameters

This function takes the following parameters:

`setFunc`	Function pointer with which the user can define their own function for setting the cookie in the foreign domain. The implementation defines the parameters.
`args`	Pointer to a pointer to agent defined parameters.

Returns

This function returns no values.

am_web_method_num_to_str()

Converts a `am_web_req_method_t` number to a string.

Details

This function is used for logging the method in the local debug logs. It takes in a method name and returns a string value (such as GET, or POST).

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT const char *
am_web_method_num_to_str(am_web_req_method_t method);
```

Parameters

This function takes the following parameter:

`method` One of the following values of the `am_web_req_method_t` enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_WEB_REQUEST_UNKNOWN,
    AM_WEB_REQUEST_GET,
    AM_WEB_REQUEST_POST,
    AM_WEB_REQUEST_HEAD,
    AM_WEB_REQUEST_PUT,
    AM_WEB_REQUEST_DELETE,
    AM_WEB_REQUEST_TRACE,
    AM_WEB_REQUEST_OPTIONS
} am_web_req_method_t;
```

More information on these request methods can be found in `http://www.faqs.org/rfcs/rfc2068.html`.

Returns

This function returns a pointer to the string. If the number passed is not recognized, `UNKNOWN` is returned.

am_web_method_str_to_num()

Converts a `am_web_req_method_t` string to a number.

Details

This function does the opposite of the previously defined `am_web_method_num_to_str()`. See "`am_web_method_num_to_str()`" on page 195 for details.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT const char *
am_web_method_num_to_str(am_web_req_method_t method);
```

Parameters

This function takes the following parameter:

`method` One of the following values of the `am_web_req_method_t` enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_WEB_REQUEST_UNKNOWN,
    AM_WEB_REQUEST_GET,
    AM_WEB_REQUEST_POST,
    AM_WEB_REQUEST_HEAD,
    AM_WEB_REQUEST_PUT,
    AM_WEB_REQUEST_DELETE,
    AM_WEB_REQUEST_TRACE,
    AM_WEB_REQUEST_OPTIONS
} am_web_req_method_t;
```

More information on these request methods can be found in http://www.faqs.org/rfcs/rfc2068.html.

Returns

This function returns a pointer to the number If the string is not recognized, AM_WEB_REQUEST_UNKNOWN is returned.

am_web_postcache_data_cleanup()

Cleans up am_web_postcache_data_t data type.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_postcache_data_cleanup(am_web_postcache_data_t * const postentry_struct);
```

Parameters

This function takes the following parameter:

postentry_struct	Pointer to am_web_postcache_data_t data type.

Returns

This function returns no value.

am_web_postcache_insert()

Inserts POST data entry in the POST cache.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_postcache_insert(const char *key,
                        const am_web_postcache_data_t *value);
```

Parameters

This function takes the following parameters:

`key`	Pointer to the POST data preservation key for every entry.
`value`	Pointer to the `am_web_postcache_data_t` data type.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the `<am_types.h>` header file):

`B_TRUE`	If the insertion was successful.
`B_FALSE`	Otherwise.

am_web_postcache_lookup()

Looks up data in the POST cache.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_postcache_lookup(const char *key,
                        am_web_postcache_data_t *postdata_entry);
```

Parameters

This function takes the following parameters:

`key`	Pointer to the key to search POST data entry in POST data structure.
`postdata_entry`	Pointer to the `am_web_postcache_data_t` data type storing the POST data.

Returns

This function returns one of the following values of the `boolean_t` enumeration (defined in the `<am_types.h>` header file):

`B_TRUE` If the search was successful.

`B_FALSE` Otherwise.

am_web_postcache_remove()

Removes data from the POST cache.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_postcache_remove(const char *key);
```

Parameters

This function takes the following parameter:

`key` Pointer to the key of the entry to be removed.

Returns

This function returns no value.

am_web_process_request()

Processes a request access check and returns a HTTP result to be rendered by the agent.

Details

The render status is returned in the `render_sts` argument.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_web_result_t
am_web_process_request(am_web_request_params_t *req_params,
                       am_web_request_func_t *req_func,
                       am_status_t *render_sts);
```

Parameters

This function takes the following parameters:

`req_params` Pointer to a `am_web_request_params_t` data type.

`req_func` Pointer to a `am_web_request_func_t` data type.

`render_sts` Pointer to one of the values of the `am_status_t` enumeration as defined in the `<am_types.h>` header file.

Returns

One of the following values of the `am_web_result_t` enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_WEB_RESULT_OK,               /* access check was OK */
    AM_WEB_RESULT_OK_DONE,          /* OK and handled (for ex. notification) */
    AM_WEB_RESULT_FORBIDDEN,        /* access forbidden */
    AM_WEB_RESULT_REDIRECT,         /* redirected */
    AM_WEB_RESULT_ERROR             /* internal error */
} am_web_result_t;
```

am_web_remove_authnrequest()

Removes those extra parameters from an authenticated request.

Details

When a user returns from CDSSO authentication, the request contains a list of parameters that were added when the user was authenticated. `am_web_remove_authnrequest()` removes these extra parameters so the request is forwarded to applications as it originally came from the browser.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_remove_authnrequest(char* inpString,
                           char **outString );
```

Parameters

This function takes the following parameters:

`inpString` Pointer to the URL received after CDSSO authentication.

`outString` Pointer to a pointer to the location of the new URL without the parameters.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If successful.
`AM_*`	The type of error indicated by the status value.

Memory Concerns

The value returned should be freed using `am_web_free()`.

am_web_remove_parameter_from_query()

Removes the given query parameter from the URL, if present.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_remove_parameter_from_query(const char* inpString,
                                   const char *remove_str,
                                   char **outString );
```

Parameters

This function takes the following parameters:

`inpString`	Pointer to the original URL.
`remove_str`	Pointer to the query parameter to be removed
`outString`	Pointer to a pointer to the location of the new URL without the query parameter.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If the call was successful.
`AM_*`	If any error occurs, the type of error indicated by the status value.

Memory Concerns

The value returned should be freed using `am_web_free()`.

am_web_result_attr_map_set()

Processes attr_response_map from a am_policy_result_t data type and performs the set action.

Details

This function replaces am_web_do_result_attr_map_set() which is deprecated. It needs to be explicitly declare to use it. See <am_web.h> for more information.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_result_attr_map_set(am_policy_result_t *result,
                           am_web_result_set_header_func_t setHeaderFunc,
                           am_web_result_set_header_attr_in_response_func_t setCookieRespFunc,
                           am_web_result_set_header_attr_in_request_func_t setCookieReqFunc,
                           am_web_get_cookie_sync_func_t getCookieSyncFunc,
                           void **args);
```

Parameters

This function takes the following parameters:

result — Pointer to the am_policy_result_t.

Note – See "am_policy_result_t" on page 47 for more information.

setHeaderFunc — Pointer to the am_web_result_set_header_func_t.

setCookieRespFunc — Pointer to the am_web_result_set_header_attr_in_response_func_t.

setCookieReqFunc — Pointer to the am_web_result_set_header_attr_in_request_func_t.

getCookieSyncFunc — Pointer to the am_web_get_cookie_sync_func_t.

args — Pointer to a pointer to agent defined parameters.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS — If the call was successful.

AM_* — If any error occurs, the type of error indicated by the status value.

Memory Concerns

The value returned should be freed using `am_web_free()`.

am_web_result_num_to_str()

Returns the name of a `am_web_result_t` as a string.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT const char *
am_web_result_num_to_str(am_web_result_t result);
```

Parameters

This function takes the following parameter:

`result` One of the following values of the `am_web_result_t` enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_WEB_RESULT_OK,               /* access check was OK */
    AM_WEB_RESULT_OK_DONE,          /* OK and handled (for ex. notification) */
    AM_WEB_RESULT_FORBIDDEN,        /* access forbidden */
    AM_WEB_RESULT_REDIRECT,         /* redirected */
    AM_WEB_RESULT_ERROR             /* internal error */
} am_web_result_t;
```

Returns

This function returns a pointer to the string. For example, `AM_WEB_RESULT_OK` returns `AM_WEB_RESULT_OK`. If the result code passed is not recognized, `Unknown result code` is returned.

am_web_set_cookie()

Sets the specified cookie in the cookie header of the request.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_set_cookie(char *cookie_header,
                  const char *set_cookie_value,
                  char **new_cookie_header);
```

Parameters

This function takes the following parameters:

`cookie_header`	Pointer to the cookie header.
`set_cookie_value`	Pointer to the cookie name and value in the *set-cookie response header* form. This value should be the same as that of the `cookieValues` parameter of the `am_web_result_set_header_attr_in_request_func_t` function.
`new_cookie_header`	Pointer to a pointer to the original cookie header, or a new cookie header value which needs to be freed by the caller. This value can be `NULL`.

Returns

This function returns one of the following values of the `am_status_t` enumeration (defined in the `<am_types.h>` header file):

`AM_SUCCESS`	If successful.
`AM_*`	The type of error indicated by the status value.

CHAPTER 9

Additional Data Types and Functions

This chapter provides information and a reference guide to the data types and functions not documented elsewhere. This includes those described in the `<am.h>`, `<am_notify.h>`, `<am_string_set.h>`, `<am_types.h>`, and `<am_util.h>` header files. Reference summaries include a short description, syntax, parameters and returns. This chapter contains the following sections:

<am.h>

`<am.h>` contains the `am_cleanup()` function which cleans up all internal data structures created by `am_sso_init()`, `am_auth_init()`, or `am_policy_init()`. It needs to be called only once at the end of any calls. After cleanup, the relevant initialize function must be called again before using any of its interfaces.

Note – Any properties passed to the initialization functions `am_sso_init()`, `am_auth_init()`, or `am_policy_init()` should be destroyed only after `am_cleanup()` is called.

---

am_cleanup() Syntax

```
#include "am.h"
AM_EXPORT am_status_t
am_cleanup(void);
```

am_cleanup() Parameters

This function takes no parameters.

am_cleanup() Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If successfully cleaned up.
AM_NSPR_ERROR	Netscape Portable Runtime (NSPR) error.
AM_FAILURE	If any other error occurred.

<am_notify.h>

<am_notify.h> contains the am_notify() function which parses and processes a session or policy notification message as an XML string. If the message is a session notification, any token handle listeners registered using am_sso_add_listener() will be called. If the message is a policy notification, the internal policy cache maintained by the policy module in the C SDK will be updated with the notification information only if the policy module has been initialized (using am_policy_init() and am_policy_service_init()).

am_notify() Syntax

```
#include "am_notify.h"
AM_EXPORT am_status_t
am_notify(const char *xmlmsg,
          am_policy_t policy_handle);
```

am_notify() Parameters

This function takes the following parameters:

xmlmsg	Pointer to the XML message containing the notification.
policy_handle	Reference to the policy evaluation object.

am_notify() Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the XML was successfully parsed and processed.
AM_INVALID_ARGUMENT	If any input parameter is invalid.
AM_ERROR_PARSING_XML	If an error occurred parsing the XML.
AM_ERROR_DISPATCH_LISTENER	If an error occurred dispatching the listener.
AM_FAILURE	If any other error occurred.

<am_string_set.h>

<am_string_set.h> contains public data types and functions intended to manipulate strings. The sample sources (located in the /*AccessManager-base*/SUNWam/samples/csdk directory) demonstrate basic usage of the string set API. This chapter contains the following sections:

- "String Data Types" on page 207
- "String Functions" on page 208

String Data Types

The string data type defined in <am_string_set.h> is am_string_set_t. The type holds a set of strings.

Details

am_string_set_allocate() and am_string_set_destroy() are used to allocate and free space for this type.

Syntax

```
#include "am_string_set.h"
typedef struct {
    int size;
    char **strings;
} am_string_set_t;
```

Members

am_string_set_t has the following members:

`size`	Number of strings
`strings`	Pointer to a pointer to a list of strings.

String Functions

The string functions defined in `<am_string_set.h>` are:

- "`am_string_set_allocate()`" on page 208
- "`am_string_set_destroy()`" on page 208

am_string_set_allocate()

Allocates memory and initializes the string set size.

Syntax

```
#include "am_string_set.h"
AM_EXPORT am_string_set_t *
am_string_set_allocate(int size);
```

Parameters

This function takes the following parameter:

`size`	Number of strings in the set.

Returns

This function returns the allocated `am_string_set_t`, or `NULL` if size is less than zero.

am_string_set_destroy()

Releases memory in the specified string set object by freeing each string in the set, then freeing the associated pointers, and finally, the structure itself.

Syntax

```
#include "am_string_set.h"
AM_EXPORT void
am_string_set_destroy(am_string_set_t *string_set);
```

Parameters

This function takes the following parameter:

`string_set` Pointer to the specified string set object.

Returns

This function returns no values.

<am_types.h>

<am_types.h> contains defined types and corresponding functions. They are:

- "boolean_t" on page 209
- "bool_t" on page 209
- "am_status_t" on page 209
- "am_status_to_string()" on page 210

boolean_t

Defines true or false boolean with the syntax:

```
#include "am_types.h"
typedef enum {
    B_FALSE,
    B_TRUE
} boolean_t;
```

bool_t

Defines true or false boolean with the syntax:

```
#include "am_types.h"
typedef enum {
    AM_FALSE = 0,
    AM_TRUE
} am_bool_t;
```

am_status_t

Defines error codes with the syntax:

```
#include "am_types.h"
typedef enum {
```

```
    AM_SUCCESS = 0,
    AM_FAILURE,
    AM_INIT_FAILURE,
    AM_AUTH_FAILURE,
    AM_NAMING_FAILURE,
    AM_SESSION_FAILURE,
    AM_POLICY_FAILURE,
    AM_NO_POLICY,
    AM_INVALID_ARGUMENT,
    AM_INVALID_VALUE,
    AM_NOT_FOUND,
    AM_NO_MEMORY,
    AM_NSPR_ERROR,
    AM_END_OF_FILE,
    AM_BUFFER_TOO_SMALL,
    AM_NO_SUCH_SERVICE_TYPE,
    AM_SERVICE_NOT_AVAILABLE,
    AM_ERROR_PARSING_XML,
    AM_INVALID_SESSION,
    AM_INVALID_ACTION_TYPE,
    AM_ACCESS_DENIED,
    AM_HTTP_ERROR,
    AM_INVALID_FQDN_ACCESS,
    AM_FEATURE_UNSUPPORTED,
    AM_AUTH_CTX_INIT_FAILURE,
    AM_SERVICE_NOT_INITIALIZED,
    AM_INVALID_RESOURCE_FORMAT,
    AM_NOTIF_NOT_ENABLED,
    AM_ERROR_DISPATCH_LISTENER,
    AM_REMOTE_LOG_FAILURE,
    AM_LOG_FAILURE,
    AM_REMOTE_LOG_NOT_INITIALIZED,
    AM_NUM_ERROR_CODES    /* This should always be the last. */
} am_status_t;
```

am_status_to_string()

Returns a message for the given error code.

Syntax

```
#include "am_types.h"
AM_EXPORT const char *am_status_to_string(am_status_t status);
```

Parameters

This function takes the following parameter:

`status`	Given error code

Returns

This function returns the appropriate message for the status code as a const char *

`AM_SUCCESS`	Success.
`AM_FAILURE`	Failure.
`AM_INIT_FAILURE`	Initialization failure.
`AM_AUTH_FAILURE`	Access Manager Authentication Service failure.
`AM_NAMING_FAILURE`	Access Manager Naming Service failure.
`AM_SESSION_FAILURE`	Access Manager Session Service failure.
`AM_POLICY_FAILURE`	Access Manager Policy Service failure.
`AM_NO_POLICY`	No policy found.
`AM_INVALID_ARGUMENT`	Invalid argument.
`AM_INVALID_VALUE`	Invalid value.
`AM_NOT_FOUND`	Access Manager not found.
`AM_NO_MEMORY`	No memory.
`AM_NSPR_ERROR`	NSPR error.
`AM_END_OF_FILE`	Reached end of file.
`AM_BUFFER_TOO_SMALL`	If the defined size of the buffer is smaller than the encoded value.
`AM_NO_SUCH_SERVICE_TYPE`	No such service type found.
`AM_SERVICE_NOT_AVAILABLE`	Service is not available.
`AM_ERROR_PARSING_XML`	Error found during XML parsing.
`XML AM_INVALID_SESSION`	Invalid session.
`AM_END_OF_FILE`	Reached end of file.
`AM_INVALID_ACTION_TYPE`	Invalid action type.
`AM_ACCESS_DENIED`	Access denied.
`AM_HTTP_ERROR`	HTTP error.
`AM_INVALID_FQDN_ACCESS`	Invalid fully qualified domain name (FQDN) access.

AM_FEATURE_UNSUPPORTED	The feature or configuration is unsupported.
AM_AUTH_CTX_INIT_FAILURE	Authentication context initialization failed.
AM_SERVICE_NOT_INITIALIZED	Specified service was not initialized.
AM_INVALID_RESOURCE_FORMAT	Specified resource name does not follow the format required by the service.
AM_NOTIF_NOT_ENABLED	Notification Service is not enabled or no notification URL is set.
AM_ERROR_DISPATCH_LISTENER	Error occurred dispatching single sign-on (SSO) listener.
AM_REMOTE_LOG_FAILURE	Remote Logging Service encountered an error.
AM_LOG_FAILURE	Log encountered an error.
AM_REMOTE_LOG_NOT_INITIALIZED	Remote Logging Service is not initialized.

<am_utils.h>

<am_utils.h> contains functions to encode and decode cookies. The functions are:

- "am_http_cookie_encode()" on page 212
- "am_http_cookie_decode()" on page 213

am_http_cookie_encode()

Encodes an HTTP cookie.

Syntax

```
#include "am.h"
AM_EXPORT am_status_t
am_http_cookie_encode(const char *cookie,
                      char *buf,
                      int len);
```

Parameters

This function takes the following parameters:

cookie	Pointer to the cookie.
buf	Pointer to the buffer where the encoded cookie will be stored.
len	The size of the buffer.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the cookie was successfully encoded and stored.
AM_INVALID_ARGUMENT	If the cookie or buffer is NULL..
AM_BUFFER_TOO_SMALL	If the defined size is smaller than the encoded value.
AM_FAILURE	If any other error occurred.

am_http_cookie_decode()

Decodes an HTTP cookie.

Syntax

```
#include "am.h"
AM_EXPORT am_status_t
am_http_cookie_decode(const char *cookie,
                      char *buf,
                      int len);
```

Parameters

This function takes the following parameters:

cookie	Pointer to the cookie.
buf	Pointer to the buffer where the encoded cookie will be stored.
len	The size of the buffer.

Returns

This function returns one of the following values of the am_status_t enumeration (defined in the <am_types.h> header file):

AM_SUCCESS	If the cookie was successfully decoded and coped.
AM_INVALID_ARGUMENT	If the cookie or buffer is NULL..
AM_BUFFER_TOO_SMALL	If the defined size is smaller than the decoded value.
AM_FAILURE	If any other error occurred.