Sun Java™ System

# Access Manager
# Developer's Reference

2005Q1

# Contents

# About This Guide

This *Developer's Reference* provides summaries of data types, structures, and functions that make up the public C APIs for Sun Java™ System Access Manager 6 2005Q1 (formerly Sun™ ONE Identity Server). Refer to it as you develop new plug-ins for use with Access Manager. Note that you will find the Javadocs for Access Manager Java APIs in this location:

*AccessManager-base*/SUNWam/docs/am_public_javadocs.jar

This preface includes the following topics:

# Audience for This Guide

This *Developer's Reference* is intended for use by IT administrators and software developers who implement an integrated identity management and web access platform using Sun Java System servers and software. It is recommended that administrators understand the following technologies:

- Lightweight Directory Access Protocol (LDAP)

- Java™ technology

- JavaServer Pages™ (JSP) technology

- HyperText Transfer Protocol (HTTP)

- HyperText Markup Language (HTML)

- eXtensible Markup Language (XML)

Because Sun Java System Directory Server is used as the data store in an Access Manager deployment, administrators should also be familiar with the documentation provided with that product. The latest Directory Server documentation can be accessed online.

# Before You Read This Guide

Access Manager is a component of the Sun Java Enterprise System, a software infrastructure that supports enterprise applications distributed across a network or Internet environment. You should be familiar with the documentation provided with Sun Java Enterprise System, which you can access online at:

http://docs.sun.com/prod/entsys.05q1

Because Sun Java System Directory Server is used as the data store in an Access Manager deployment, you should be familiar with the Directory Server documentation, which you can access online at:

http://docs.sun.com/coll/DirectoryServer_05q1

# Conventions Used in This Guide

The tables in this section describe the conventions used in this guide.

## Typographic Conventions

The following table describes the typographic changes used in this guide.

**Table 1**    Typographic Conventions

| Typeface | Meaning | Examples |
|---|---|---|
| AaBbCc123 (Monospace) | API and language elements, HTML tags, web site URLs, command names, file names, directory path names, onscreen computer output, sample code. | Edit your .login file. Use ls -a to list all files. % You have mail. |

**Table 1**    Typographic Conventions *(Continued)*

| Typeface | Meaning | Examples |
|---|---|---|
| **AaBbCc123** (Monospace bold) | What you type, when contrasted with onscreen computer output. | % **su**<br>Password: |
| *AaBbCc123* (Italic) | Book titles, new terms, words to be emphasized. | Read Chapter 6 in the *User's Guide*. |
| | A placeholder in a command or path name to be replaced with a real name or value. | These are called *class* options.<br><br>Do *not* save the file.<br><br>The file is located in the *install-dir*/bin directory. |

# Symbols

The following table describes the symbol conventions used in this guide.

**Table 2**    Symbol Conventions

| Symbol | Description | Example | Meaning |
|---|---|---|---|
| [ ] | Contains optional command options. | ls [-l] | The -l option is not required. |
| { \| } | Contains a set of choices for a required command option. | -d {y\|n} | The -d option requires that you use either the y argument or the n argument. |
| - | Joins simultaneous multiple keystrokes. | Control-A | Press the Control key while you press the A key. |
| + | Joins consecutive multiple keystrokes. | Ctrl+A+N | Press the Control key, release it, and then press the subsequent keys. |
| > | Indicates menu item selection in a graphical user interface. | File > New > Templates | From the File menu, choose New. From the New submenu, choose Templates. |

# Default Paths and File Names

The following table describes the default paths and file names used in this guide:

**Table 3**     Default Paths and File Names

| Term | Description |
|------|-------------|
| *AccessManager-base* | Represents the base installation directory for Access Manager. The Access Manager default base installation and product directory depends on your specific platform: <br><br>Solaris™ systems: `/opt/SUNWam` <br><br>Linux systems: `/opt/sun/identity` |
| *DirectoryServer-base* | Represents the base installation directory for Sun Java System Directory Server. Refer to the product documentation for the specific path name. |
| *ApplicationServer-base* | Represents the base installation directory for Sun Java System Application Server. Refer to the product documentation for the specific path name. |
| *WebServer-base* | Represents the base installation directory for Sun Java System Web Server. Refer to the product documentation for the specific path name. |

# Shell Prompts

The following table describes the shell prompts used in this guide.

**Table 4**     Shell Prompts

| Shell | Prompt |
|-------|--------|
| C shell on UNIX or Linux | *machine-name*`%` |
| C shell superuser on UNIX or Linux | *machine-name*`#` |
| Bourne shell and Korn shell on UNIX or Linux | `$` |
| Bourne shell and Korn shell superuser on UNIX or Linux | `#` |
| Windows command line | `C:\` |

# Related Documentation

To access Sun technical documentation online, go to `http://docs.sun.com`.

You can browse the documentation archive or search for a specific book title, part number, or subject.

## Books in This Documentation Set

**Table 5**    Access Manager 6 2005Q1 Documentation Set

| Title | Description |
|---|---|
| *Technical Overview*<br>`http://docs.sun.com/doc/817-7643` | Provides a high-level overview of how Access Manager components work together to consolidate identity management and to protect enterprise assets and web-based applications. Explains basic Access Manager concepts and terminology |
| *Deployment Planning Guide*<br>`http://docs.sun.com/doc/817-7644` | Provides information about planning a deployment within an existing information technology infrastructure |
| *Administration Guide*<br>`http://docs.sun.com/doc/817-7647` | Describes how to use the Access Manager console as well as manage user and service data via the command line. |
| *Migration Guide*<br>`http://docs.sun.com/doc/817-7645` | Describes how to migrate existing data and Sun Java System product deployments to the latest version of Access Manager. (For instructions about installing and upgrading Access Manager and other products, see the *Sun Java Enterprise System 2005Q1 Installation Guide*.) |
| *Performance Tuning Guide*<br>`http://docs.sun.com/doc/817-7646` | Describes how to tune Access Manager and its related components. |
| *Federation Management Guide*<br>`http://docs.sun.com/doc/817-7648` | Provides information about Federation Management, which is based on the Liberty Alliance Project. |
| *Developer's Guide*<br>`http://docs.sun.com/doc/817-7649` | Offers information on how to customize Access Manager and integrate its functionality into an organization's current technical infrastructure. Contains details about the programmatic aspects of the product and its API. |
| *Developer's Reference*<br>`http://docs.sun.com/doc/817-7650` | Provides summaries of data types, structures, and functions that make up the Access Manager public C APIs. |

**Table 5**    Access Manager 6 2005Q1 Documentation Set *(Continued)*

| Title | Description |
| --- | --- |
| *Release Notes*<br><br>http://docs.sun.com/doc/817-7642 | Available after the product is released. Contains 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. |

# Access Manager Policy Agent Documentation

Documentation for the Access Manager Policy Agents is available on the following documentation Web site:

http://docs.sun.com/coll/S1_IdServPolicyAgent_21

Policy Agents for Access Manager are available on a different schedule than the server product itself. Therefore, the documentation set for the policy agents is available outside the core set of Access Manager documentation. The following titles are included in the set:

- *Policy Agents For Web and Proxy Servers Guide* documents how to install and configure an Access Manager policy agent on various web and proxy servers. It also includes troubleshooting and information specific to each agent.

- *J2EE Policy Agents Guide* documents how to install and configure an Access Manager policy agent that can protect a variety of hosted J2EE applications. It also includes troubleshooting and information specific to each agent.

- The *Release Notes* are available online after a set of agents is released. The *Release Notes* include a description of what is new in the current release, known problems and limitations, installation notes, and how to report issues with the software or the documentation.

## Other Server Documentation

For other server documentation, go to the following:

- Directory Server documentation
  http://docs.sun.com/coll/DirectoryServer_05q1

- Web Server documentation
  http://docs.sun.com/coll/WebServer_05q1

- Application Server documentation
  http://docs.sun.com/coll/ApplicationServer8_ee_04q4

- Web Proxy Server documentation
  http://docs.sun.com/prod/s1.webproxys#hic

# Accessing Sun Resources Online

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

Download Center
http://wwws.sun.com/software/download/

Sun Java System Services Suite
http://www.sun.com/service/sunps/sunone/index.html

Sun Enterprise Services, Solaris Patches, and Support
http://sunsolve.sun.com/

Developer Information
http://developers.sun.com/prodtech/index.html

# Contacting Sun Technical Support

If you have technical questions about this product that are not answered in the product documentation, go to:

http://www.sun.com/service/contacting.

# Related Third-Party Web Site References

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 or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

# Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions.

To share your comments, go to `http://docs.sun.com` and click Send Comments. In the online form, provide 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 guide or at the top of the document.

For example, the title of this guide is *Sun Java System Access Manager 6 2005Q1 Developer's Reference*, and the part number is 817-7650.

# Type and Structure Reference

This chapter covers the types and structures provided in the C SDK available for use to interact with Sun Java™ System Access Manager 6 2005Q1. All authentication related types and structures can be found in the C SDK include file `am_auth.h`. The following structures are summarized in this chapter:

## am_auth_callback

Primary callback structure for authentication.

This structure is a C implementation of the Java 2 SDK `javax.security.auth.callback` interface used to submit authentication requirements to the authentication service on the Access Manager. The Access Manager authentication service framework is based on the Java 2 SDK JAAS 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_tconfirmation_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;
```

**Fields**

This structure has the following fields:

| Field | Description |
|---|---|
| callback_type | Indicates which type of callback this represents and determines which callback structure is used in the callback_info union below. |
| | The value is one of the following: |
| | • ChoiceCallback |
| | • ConfirmationCallback |
| | • LanguageCallback, NameCallback |
| | • TextInputCallback |
| | • TextOutputCallback |
| | Each callback type corresponds to the callback class of the same name in the Java 2 SDK javax.security.auth.callback package. |
| callback_info | The union of possible callback structures. The structure in the union to use depends on the callback_type field. Each structure corresponds to the callback class of the same name in the Java 2 SDK javax.security.auth.callback package and, has a response field to submit callback requirements. |
| | Note that memory for all fields in the callback structures except the response field is allocated by the C SDK in the am_auth_login() call, and is freed by the C SDK when the auth context is destroyed using am_auth_destroy_auth_context(). Memory for the response field must be allocated and freed by the caller. |
| | Each callback structure is described in this chapter in detail. |

# am_auth_choice_callback

Choice authentication callback structure.

This is a C implementation of the javax.security.auth.callback.ChoiceCallback class used to submit authentication callback requirements to the Access Manager Authentication service.

**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;
```

**Fields**

This structure should be used if the `callback_type` is `ChoiceCallback` used to submit authentication callback requirements to the Access Manager authentication service.

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

It has the following fields:

| Field | Description |
|---|---|
| prompt | Prompt to describe the list of choices. |
| allow_multiple_selections | True if this choice allows multiple selections. |
| choices | Choices for this choice callback. The number of choices is indicated in the `choices_size` field. Memory for choices list is allocated by the C SDK in `am_auth_login()` and is freed by the C SDK when the authentication context is destroyed using `am_auth_destroy_auth_context()`. |
| choices_size | Number of choices in the choices field. |
| default_choice | Default choice, as an index into the choices list. |
| response | Selected choices. |
| | Memory for the response must be allocated and freed by the caller. |
| response_size | The number of selected choices in the response. |

**Details**

See `am_auth_test.c` in the C SDK samples for an example of how to use the choice callback.

# am_auth_confirmation_callback_info

Confirmation authentication callback structure.

This is a C implementation of the
`javax.security.auth.callback.ConfirmationCallback` class used to submit
authentication callback requirements to the Access Manager authentication service.

**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;
```

**Fields**

This structure has the following fields:

| Field | Description |
| --- | --- |
| prompt | prompt to describe the options, if any. |
| message_type | The message type: "INFORMATION", "WARNING" or "ERROR". |
| | Memory for the message type is allocated by the C SDK in `am_auth_login()` and freed when the authentication context is destroyed using `am_auth_destroy_auth_context()`. |
| option_type | The option type: "YES_NO_OPTION", "YES_NO_CANCEL_OPTION", "OK_CANCEL_OPTION", or "UNSPECIFIED". |
| | Memory for the message type is allocated by the C SDK in `am_auth_login()` and freed when the authentication context is destroyed using `am_auth_destroy_auth_context()`. |
| options | The list of confirmation options, or null if this `ConfirmationCallback` was instantiated with an optionType instead of options. |
| | Memory for the options list is allocated by the C SDK in `am_auth_login()` and freed when the authentication context is destroyed using `am_auth_destroy_auth_context()`. |
| options_size | Number options in the options list. |
| default_option | The default option, if any. |
| | Memory for the default option is allocated by the C SDK in `am_auth_login()` and freed when the authentication context is destroyed using `am_auth_destroy_auth_context()`. |

| | |
|---|---|
| response | The selected option. |
| | Memory for the response must be allocated and freed by the caller. |

### Details

See `am_auth_test.c` in the C SDK samples for an example of how to use the confirmation callback.

# am_auth_language_callback_info

Language callback structure.

### 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;
```

### Fields

This structure has the following fields:

| Field | Description |
|---|---|
| locale | The locale from Access Manager. |
| | Memory for the locale is allocated by the C SDK in `am_auth_login()` and freed when the authentication context is destroyed using `am_auth_destroy_auth_context()`. |
| response | The locale to send back to Access Manager. |
| | Memory for the response must be allocated and freed by the caller. |

# am_auth_locale

Language locale structure.

**Syntax**

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

**Fields**

This structure has the following fields:

| Field | Description |
| --- | --- |
| language | A valid ISO Language Code. These codes are the lower-case, two-letter codes as defined by ISO-639. You can find a full list of these codes at a number of sites, such as: |
| | `http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt` |
| country | A valid ISO Country Code. These codes are the upper-case, two-letter codes as defined by ISO-3166. You can find a full list of these codes at a number of sites, such as: |
| | `http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html` |
| variant | A vendor or browser-specific code. For example, WIN for Windows, MAC for Macintosh, and POSIX for POSIX. |

**Details**

See `am_auth_test.c` in the C SDK samples for an example of how to use this structure with the locale callback.

# am_auth_name_callback_info

Name callback structure.

This is a C implementation of the `javax.security.auth.callback.NameCallback` class used to submit authentication callback requirements to the Access Manager authentication service.

**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;
```

**Fields**

This structure has the following fields:

| Field | Description |
| --- | --- |
| prompt | Prompt for the name, if any. |
| | Memory for the prompt is allocated by the C SDK in am_auth_login() and freed when the authentication context is destroyed using am_auth_destroy_auth_context(). |
| default_name | Default name, if any. |
| | Memory for the default name is allocated by the C SDK in am_auth_login() and freed when the authentication context is destroyed using am_auth_destroy_auth_context(). |
| response | The name to be submitted to the Access Manager. |
| | Memory for the response must be allocated and freed by the caller. |

**Details**

See am_auth_test.c in the C SDK samples for an example of how to use the name callback.

# am_auth_password_callback_info

Password callback structure.

This is a C implementation of the javax.security.auth.callback.PasswordCallback class used to submit authentication callback requirements to the Access Manager authentication service.

**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;
```

**Fields**
This structure has the following fields:

| Field | Description |
|---|---|
| prompt | Prompt for the password, if any. |
| | Memory for the prompt is allocated by the C SDK in am_auth_login() and freed when the authentication context is destroyed using am_auth_destroy_auth_context(). |
| echo_on | Whether the password should be displayed as it is typed. |
| response | The password to be submitted to Access Manager. |
| | Memory for the response must be allocated and freed by the caller. |

**Details**
See am_auth_test.c in the C SDK samples for an example of how to use the
password callback.

# am_auth_text_input_callback_info

Text Input authentication callback structure.

This is a C implementation of the
javax.security.auth.callback.TextInputCallback class used to submit
authentication callback requirements to the Access Manager authentication service.

**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;
```

**Fields**

This structure has the following fields:

| Field | Description |
| --- | --- |
| prompt | Prompt for the text input, if any. |
| | Memory for the prompt is allocated by the C SDK in am_auth_login() and freed when the authentication context is destroyed using am_auth_destroy_auth_context(). |
| default_text | Default text for the text input, if any. |
| | Memory for the default text is allocated by the C SDK in am_auth_login() and freed when the authentication context is destroyed using am_auth_destroy_auth_context(). |
| response | Text input to be submitted to the Access Manager. |
| | Memory for the response must be allocated and freed by the caller. |

**Details**

See am_auth_test.c in the C SDK samples for an example of how to use the password callback.

# am_auth_text_output_callback_info

Text Output callback structure.

**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;
```

**Fields**

This structure has the following fields:

| Field | Description |
| --- | --- |
| message | Message to be displayed. |
| | Memory for the message is allocated by the C SDK in am_auth_login() and freed when the authentication context is destroyed using am_auth_destroy_auth_context(). |

| | |
|---|---|
| message_type | Message type, one of "INFORMATION", "WARNING" or "ERROR". |
| | Memory for the message type is allocated by the C SDK in am_auth_login() and freed when the authentication context is destroyed using am_auth_destroy_auth_context(). |

**Details**

See am_auth_test.c in the C SDK samples for an example of how to use the text output callback.

# am_log_record

Log Record

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

**Fields**
This is an opaque structure and therefore has no fields accessible by the C SDK user.

**Details**
See am_log_test.c in the C SDK samples for an example of how to use the text output callback.

# am_map_t

Opaque handle to a map object. A map object is used to manipulate key value pairs using the am_map_* interface. Map objects are used by the policy interface in the C SDK to return any policy decision results and advices from Access Manager policy service, and to pass any environment variables for to the policy interface for policy evaluation.

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

**Fields**

This is an opaque structure and therefore has no fields accessible by the C SDK user.

**Details**

See `am_policy_test.c` in the C SDK samples for an example of how to use `am_map_t`.

# am_map_entry_iter

Opaque handle 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;
```

**Fields**

This is an opaque structure and therefore has no fields accessible by the C SDK user.

**Details**

See `am_policy_test.c` in the C SDK samples for an example of how to use `am_map_entry_iter`.

# am_map_value_iter

Opaque handle to an iterator for the entries in a map object `am_map_t`. A map object is used to manipulate key value pairs using the `am_map_* interface`. Map objects are used by the policy interface in the C SDK to return any policy decision results and advices from Access Manager policy service, and to pass any environment variables for policy evaluation.

**Syntax**

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

**Fields**

This is an opaque structure and therefore has no fields accessible by the C SDK user.

**Details**

See am_policy_test.c in the C SDK samples for an example of how to use am_map_entry_iter_t.

# am_policy_result

Policy evaulation results from the policy interface in the C SDK.

Memory for am_policy_result is allocated by am_policy_evaluate() in the C SDK and should be freed by calling am_policy_result_destroy().

**Syntax**

```
#include "am_policy.h"
typedef struct am_policy_result {
    const char *remote_user;
    const char *remote_IP;

    am_map_t advice_map;
    am_map_t attr_response_map;
} am_policy_result_t;
```

**Fields**

This structure has the following fields:

| Field | Description |
| --- | --- |
| remote_use | The remote user. |
| remote_IP | The remote IP. |
| advice_map | Any policy advices |
| attr_response_map | Any user attributes. |

**Details**

See am_policy_test.c in the C SDK samples for an example of how to use am_policy_result_t in the policy interfaces.

# am_properties_iter

am_resource_traits

Structure for traits of policy resources (such as URLs) to be evaluated.

The traits are used by the policy interfaces in the C SDK to determine how to compare and canonicalize policy resources to reach a policy decision during policy evaluation.

**Syntax**
```
#include "am_policy.h"
typedef struct am_resource_traits {
    am_resource_match_t (*cmp_func_ptr)(const struct am_resource_traits
v*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;
```

**Fields**

This structure has the following fields:

| Field | Description |
|---|---|
| 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); | A function that compares the policy_res_name and resource_name and returns a resource match result.<br><br>**Inputs:**<br>rsrc_traits - the resource traits structure to use.<br>policy_res_name - name of a resource in the policy tree.<br>resource_name - name of the resource in policy evaluation.<br>use_patterns - whether to use or recognize patterns when comparing resources.<br><br>**Returns:**<br>Return one of AM_SUB_RESOURCE_MATCH, AM_EXACT_MATCH, AM_SUPER_RESOURCE_MATCH, AM_NO_MATCH, or AM_EXACT_PATTERN_MATCH.<br><br>Example:<br>am_policy_compare_urls() can be used for URL resources. |
| boolean_t (*has_patterns) ( const char *resource _name); | A function to determine whether a resource has patterns.<br><br>**Inputs:**<br>resource_name - name of the resource.<br>**Returns:**<br>true if resource_name has patterns and false otherwise.<br>Example:<br>am_policy_resource_has_patterns can be used for URL resources. |

| | |
|---|---|
| `boolean_t`<br>`    (*get_resource_root)`<br>`        ( const char`<br>`          *resource_nam,`<br>`              char`<br>`*root_resource_name,`<br>`      size_t`<br>`        buflength);` | A function to get the root of a resource.<br><br>**Inputs:**<br>Resource_name - name of the resource.<br>Root_resource_name - a buffer to contain the name of the resource root.<br>Buflength - length of the root_resource_name buffer passed to this function.<br><br>**Returns:**<br>true if the name of the resource root was successfully inserted into the given `root_resource_name` buffer, false otherwise.<br><br>**Examples:**<br>`am_policy_get_url_resource_root()` can be used for URL resources. |
| `ignore_case` | whether case should be ignored for all functions in this structure. |
| `separator` | resource seperator. For URLs '/' should be used as the seperator. |
| `void`<br><br>`(*canonicalize)`<br><br>`    (const char`<br>`*resource,`<br>`     char **c_resource);` | A function to canonicalize a resource name.<br><br>Inputs:<br>resource - the resource name.<br><br>Outputs:<br>c_resource - the canonicalized resource name. Memory for the canonicalized name must be allocated by the caller. A function to free the memory allocated for the canonicalized must be set in the str_free field. |
| `void (*str_free)`<br>`   (void`<br>`*resource_str);` | A function to free the `c_resource` `string` returned in the canonicalize function above, after policy results have been evaluated by `am_policy_evaluate()`.<br>This field cannot be set to null.<br>**Inputs:**<br>`resource_str - the` string to be freed.<br>**Examples:**<br>free() should be used if the canonicalize field is set to the `am_policy_resource_canonicalize()` function. |

**Details**

See `am_policy_test.c` in the C SDK samples for an example of how this structure is used.

# am_string_set_t

Structure for containing a set of strings used by various interfaces in the SDK.

The `am_string_set_allocate()` and `am_string_set_destroy()` interfaces can be used to allocate and free space for this structure.

**Syntax**

```
#include "am_string_set.h"
typedef struct {
    int size;
    char **strings;
} am_string_set_t;
```

**Fields**

This structure has the following fields:

| Field | Description |
|---|---|
| size | Number of strings in the strings field |
| strings | List of strings |

**Details**

See C SDK samples for examples of how this structure is used.

am_string_set_t

# Authorization Functions

This chapter provides a reference to the public functions you can use in developing custom authorization modules for Sun Java™ System Access Manager 6 2005Q1. Function summaries include a short description, syntax, parameters and returns.

The following functions are contained in the header file `am_auth.h`:

- `am_auth_abort()` **on page 38**
- `am_auth_create_auth_context()` **on page 38**
- `am_auth_destroy_auth_context()` **on page 39**
- `am_auth_get_module_instance_names()` **on page 40**
- `am_auth_get_organization_name()` **on page 41**
- `am_auth_get_sso_token_id()` **on page 41**
- `am_auth_has_more_requirements()` **on page 43**
- `am_auth_init()` **on page 43**
- `am_auth_login()` **on page 44**
- `am_auth_logout()` **on page 45**
- `am_auth_num_callbacks()` **on page 45**
- `am_auth_submit_requirements()` **on page 46**

# am_auth_abort()

Aborts the authentication process.

### 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:

| Parameter | Description |
|-----------|-------------|
| auth_ctx | Handle of the auth context. |

### Returns
This function returns `am_status_t` with one of the following values:

| Value | Description |
|-------|-------------|
| AM_SUCCESS | If the abort process was successfully completed. |
| AM_INVALID_ARGUMENT | If the `auth_ctx` parameter is NULL. |

# am_auth_create_auth_context()

Creates a new auth context and returns the handle.

### 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:

| Parameter | Description |
|-----------|-------------|
| auth_ctx | Pointer to the handle of the auth context. |

| | |
|---|---|
| org_name | Organization name to authenticate to. May be NULL to use value in property file. |
| cert_nick_name | The alias of the certificate to be used if the client is connecting securely. May be NULL in case of non-secure connection. |
| url | Service URL, for example:<br><br>`http://pride.red.iplanet.com:58080/amserver`<br><br>May be NULL, in which case the naming service URL property is used. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If auth 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()

Destroys the given auth context handle.

**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:

| Parameter | Description |
|---|---|
| auth_ctx | Handle of the auth context to be destroyed. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|

| | |
|---|---|
| AM_SUCCESS | If the auth context was successfully destroyed. |
| AM_INVALID_ARGUMENT | If the auth_ctx parameter is NULL. |

# am_auth_get_module_instance_names()

Gets the authentication module instances (or plug-ins) configured for an organization, or sub-organization name that was set during the creation of the auth context.

### 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:

| Parameter | Description |
|---|---|
| auth_ctx | Handle of the auth context. |
| module_inst_names_ptr | Address of a pointer to am_string_set_t. |

### Returns

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| 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 auth service is not initialized. |

### Details

Supply the address of a pointer to a structure of type am_string_set_t. Module instance names are returned in am_string_set_t. Free the memory allocated for this set by calling am_string_set_destroy().

Returns NULL if the number of modules configured is zero.

# am_auth_get_organization_name()

Gets 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:

| Parameter | Description |
|-----------|-------------|
| auth_ctx | Handle of the auth context. |

### Returns

This function returns const char * with one of the following values:

| Value | Description |
|-------|-------------|
| Zero terminated string representing the organization | When user successfuly logs in. |
| NULL | If there was an error or the user has not successfully logged in. |

# am_auth_get_sso_token_id()

Get the SSO token id of the authenticated user.

### 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:

| Parameter | Description |
|-----------|-------------|

| uth_ctx | Handle of the auth context. |
|---------|------------------------------|

**Returns**

This function returns const char * with one of the following values:

| Value | Description |
|-------|-------------|
| Zero terminated string representing the organization | When user successfuly logs in. |
| NULL | If there was an error or the user has not successfully logged in |

# am_auth_get_sso_token_id()

Get the SSO token id of the authenticated user.

**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:

| Parameter | Description |
|-----------|-------------|
| auth_ctx | Handle of the auth context. |

**Returns**

This function returns const char * with one of the following values:

| Value | Description |
|-------|-------------|
| Zero terminated string representing the organization. | When user successfuly logs in. |
| NULL | If there was an error or the user has not successfully logged in. |

# am_auth_has_more_requirements()

Checks to see if there are requirements to be supplied to complete the login process.

### 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:

| Parameter | Description |
|-----------|-------------|
| auth_ctx | Handle of the auth context. |

### Returns

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

| Value | Description |
|-------|-------------|
| B_TRUE | If there are more requirements. |
| B_FALSE | If there are no more requirements. |

### Details

This call is invoked after invoking the `login()` call. If there are requirements to be supplied, then the caller can retrieve and submit the requirements in the form of callbacks.

# am_auth_init()

Initializes the authentication modules.

### 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:

| Parameter | Description |
| --- | --- |
| auth_init_params | The property handle to the property file which contains the properties to initialize the authentication library. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
| --- | --- |
| 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 | If the error was due to other causes. See am_types.h. |

# am_auth_login()

Starts 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:

| Parameter | Description |
| --- | --- |
| auth_ctx | Handle of the auth context. |
| auth_idx | Index type to be used to initiate the login process. |
| value | Value corresponding to the index type. |

**Returns**

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

| Value | Description |
|-------|-------------|
| 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:

| Parameter | Description |
|-----------|-------------|
| auth_ctx | Handle of the auth context. |

**Returns**

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

| Value | Description |
|-------|-------------|
| AM_SUCCESS | If the logout process was successfully completed. |
| AM_INVALID_ARGUMENT | If the auth_ctx parameter is NULL. |

# am_auth_num_callbacks()

Gets 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:

| Parameter | Description |
|-----------|-------------|
| auth_ctx | Handle of the auth context. |

**Returns**

This function returns `size_t` a value equal to the number of callbacks.

# am_auth_submit_requirements()

Submits the responses populated in the callbacks to the server.

**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:

| Parameter | Description |
|-----------|-------------|
| auth_ctx | Handle of the auth context. |

**Returns**

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

| Value | Description |
|-------|-------------|
| 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. |

# Logging Functions

This chapter provides a reference to public functions in the C SDK for logging on the local system or on Sun Java™ System Access Manager 6 2005Q1. Function summaries include a short description, syntax, parameters and returns.

The following functions are contained in the header file `am_log.h`:

# am_log_add_module()

Adds a new module to the list of known logging modules.

### 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:

| Parameter | Description |
| --- | --- |
| name | The name to associate with the new module. |
| id_ptr | Where to store the id of the logging module. |

### Returns

This function returns am_status_t with one of the following values:

| Value | Description |
| --- | --- |
| AM_SUCCESS | If no error is detected. |
| AM_INVALID_ARGUMENT | If name or id_ptr is NULL. |
| AM_NSPR_ERROR | If unable to initialize to the logging package. |
| AM_NO_MEMORY | If unable to allocate memory for the new logging module. |
| AM_FAILURE | If any other error is detected. |

### Details

If a module of the same name already exists, then the module ID of that module is returned.

# am_log_flush_remote_log()

Flushes all the log records in the 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 am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If Flush to remote log was successful. |
| AM_* | If any error occurs, the type of error indicated by the status value. |

# am_log_init()

Initializes logging.

This must be called before using any am_log_* interfaces.

If any SSO, auth or policy initialization functions, am_sso_init(), am_auth_init(), or am_policy_init(), is called, then am_log_init() does not need to be called separately. Any of those functions will call am_log_init() internally with the same properties parameter that was used to initialize sso, auth or policy.

See the agents documentation on parameters related to logging that can be used to initialize log.

**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 parameters:

| Parameter | Description |
|---|---|
| log_init_params | Properties to initialize the log module with. |

**Returns**

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

| Value | Description |
|-------|-------------|
| 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()

Determines whether a logging message at the specified level and associated with the specified module would be emitted.

**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:

| Parameter | Description |
|-----------|-------------|
| module | The ID of the module to be examined. |
| level | The logging level to be checked. |

**Returns**

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

| Value | Description |
|-------|-------------|
| !0 | If the message would be emitted. |
| 0 | Otherwise |

# am_log_log()

Log the given message for the given module and at the given level.

**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:

| Parameter | Description |
|---|---|
| module | The ID of the module to be associated with the message. |
| level | The logging level of the message. |
| format | A printf-style format string. |

**Returns**
This function returns `boolean_t` with one of the following values.

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

**Details**
The message is emitted only if the current level of the specified module is greater than or equal to the specified level.

# am_log_log_record()

Logs the given log record to the given log_name on the Access Manager.

am_log_record_* interfaces can be used to set information in the log record.

**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);
```

Start here

**Parameters**
This function takes the following parameters:

| Parameter | Description |
|---|---|
| record | The log record. |
| log_name | The name of the log module to log the log record to |
| logged_by_token_id | A valid SSO token ID required to access the logging service on Access Manager. |

**Returns**
This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If the log operation was successful |
| AM_* | If any error occurs, the type of error indicated by the status value. |

# am_log_record_add_loginfo()

Updates the 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:

| Parameter | Description |
|---|---|

| | |
|---|---|
| record | Opaque handle to the log record. |
| log_name | The name of the log module to log the log record to. |
| logged_by_token_id | A valid SSO token ID required to access the logging service on Access Manager. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If the key and value was successfully added to the given log record. |
| AM_* | If any error occurs, the type of error indicated by the status value. |

# am_log_record_create()

Creates a log record and initializes it with the given log level and message.

**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:

| Parameter | Description |
|---|---|
| record | Opaque handle to the log record. |
| log_name | The name of the log module to log the log record to |
| logged_by_token_id | A valid SSO token ID required to access the logging service on Access Manager. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If the key and value was successfully added to the given log record. |

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

# am_log_record_destroy()

Destroys the log record returned by `am_log_record_create`.

**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 parameters:

| Parameter | Description |
|---|---|
| record | Opaque handle to the log record to destroy. |

**Returns**
This function returns `am_status_t` with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If the log record was successfully destroyed. |
| AM_* | If any error occurs, the type of error indicated by the status value. |

# am_log_record_populate()

Updates the log record with user's SSO token 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:

| Parameter | Description |
|---|---|

| | |
|---|---|
| record | Opaque handle to the log record. |
| user_token_id | A valid SSO Token ID. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If the operation was successful. |
| AM_* | If any error occurs, the type of error indicated by the status value. |

# am_log_record_set_log_level()

Convenience functions.

**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:

| Parameter | Description |
|---|---|
| record | Opaque handle to the log record. |
| log level | Log level to set in the log record. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If the log level was successfully set. |
| AM_* | If any error occurs, the type of error indicated by the status value. |

# am_log_record_set_log_message()

Convenience function.

### 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:

| Parameter | Description |
|-----------|-------------|
| record | Opaque handle to the log record. |
| message | The message to set in the log record. |

### Returns

This function returns am_status_t with one of the following values:

| Value | Description |
|-------|-------------|
| AM_SUCCESS | If the message was successfully added to the log record. |
| AM_* | If any error occurs, the type of error indicated by the status value. |

# am_log_record_set_loginfo_props()

Updates the log record with additional 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:

| Parameter | Description |
|-----------|-------------|
| record | Opaque handle to the log record. |

| log_info | Key value pairs to be set in the log record. |
|----------|----------------------------------------------|

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|-------|-------------|
| AM_SUCCESS | If log_info was successfully added. |
| AM_* | If any error occurs, the type of error indicated by the status value. |

**Details**

Sets all log info values as properties map.

The log_info is expected to have the required log info fields as key value pairs and user is expected to delete the am_properties_t pointer only when he is done with amsdk.

# am_log_set_levels_from_string()

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

**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:

| Parameter | Description |
|-----------|-------------|
| module_level_string | list of modules to set. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|-------|-------------|
| AM_SUCCESS | If no error is detected |

| AM_INVALID_ARGUMENT | If name or id_ptr is NULL |
| AM_NSPR_ERROR | If unable to initialize to the logging package |
| AM_NO_MEMORY | If unable to allocate memory for the new logging module |
| AM_FAILURE | If unable to allocate memory for the new logging module |

**Details**

The format of the string is:

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

Optional spaces may occur before and after any commas.

# 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:

| Parameter | Description |
|---|---|
| name | Name of the file in which to record logging messages. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If the logging file could be set |
| AM_NO_MEMORY | If unable to allocate memory for internal data structures |
| AM_FAILURE | If any other error is detected |

**Details**

If the specified name is NULL or empty, then logging messages will be sent to the stderr * stream.

# am_log_set_module_level()

Sets the logging level for the specified 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:

| Parameter | Description |
|-----------|-------------|
| moduleID | The ID of the module to be modified. |
| level | The new logging level for the module. |

### Returns

This function returns am_log_level_t with one of the following values:

| Value | Description |
|-------|-------------|
| The previous logging level of the module. | When the logging level is set properly. |
| LOG_NONE | If the specified module is invalid. |

# am_log_set_remote_info()

Sets information about Access Manager log service for the remote log module.

This must be called before calling `am_log_message()` with `AM_LOG_REMOTE_MODULE` as the log module.

Otherwise use `am_log_log()` with a log record and SSO token ID to log to Access Manager.

### 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:

| Parameter | Description |
|---|---|
| rem_log_url | URL of the Access Manager log service. |
| sso_token_id | The logged by SSO Token ID. |
| rem_log_name | The log name on Access Manager. |
| log_props | Properties to initialize the remote log service with. |

### Returns

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

| Value | Description |
|---|---|
| AM_SUCCESS | If the function call is successful. |
| AM_* | If an error occurs. |

# am_log_vlog()

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

**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, ...);
```

**Parameters**
This function takes the following parameters:

| Parameter | Description |
|-----------|-------------|
| module | The ID of the module to be associated with the message. |
| level | The logging level of the message. |
| format | A printf-style format string. |

**Returns**
The set of addition arguments needed by the format string either enumerated directly or passed using the standard va_list mechanism as appropriate to the call.

**Details**
The message is emitted only if the current level of the specified module is greater than or equal to the specified level.

am_log_vlog()

Chapter   4

# Map Functions

This chapter provides a reference to functions you can use for creating, destroying, and manipulating the map objects used by the Sun Java™ System Access Manager 6 2005Q1 Access Management SDK. Function summaries include a short description, syntax, parameters and returns.

The following functions are contained in the header file `am_map.h`.

- am_map_value_iter_get() on page 76

- am_map_value_iter_is_value_valid() on page 77

# am_map_clear()

Erases all of the entries in the specified map.

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

### Parameters
This function takes the following parameters:

| Parameter | Description |
|-----------|-------------|
| map | The handle for the map object to be modified. |

### Returns
This function returns am_status_t with one of the following values:

| Value | Description |
|-------|-------------|
| AM_SUCCESS | If no error was detected. |
| AM_INVALID_ARGUMENT | If the map argument is NULL. |

# am_map_copy()

Makes a copy of a 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:

| Parameter | Description |
| --- | --- |
| source_map | The handle for the map object to be destroyed. The handle may be NULL. |
| map_ptr | A pointer to where to store the handle of the new created map object. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
| --- | --- |
| AM_SUCCESS | If a 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. |

# am_map_create()

Creates a new, empty map object.

**Syntax**

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

**Parameters**

This function takes the following parameters:

| Parameter | Description |
| --- | --- |
| map_ptr | Pointer to where the handle for the new map object should be stored. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If a map was successfully created. |
| AM_NO_MEMORY | If unable to allocate memory for the map object. |
| AM_INVALID_ARGUMENT | If the map_ptr parameter is NULL. |

# am_map_destroy()

Destroys the map object referenced by the provided handle.

**Syntax**

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

**Parameters**

This function takes the following parameters:

| Parameter | Description |
|---|---|
| map | The handle for the map object to be destroyed.  The handle may be NULL. |

**Returns**

None

# am_map_entry_iter_destroy()

Destroys the entry iterator object referenced by the provided handle.

**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 parameters:

| Parameter | Description |
|-----------|-------------|
| entry_iter | The handle for the key iterator object to be destroyed. The handle may be NULL. |

**Returns**

None

# am_map_entry_iter_get_first_value()

Returns the first value of the element currently referenced by the specified 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 parameters:

| Parameter | Description |
|-----------|-------------|
| entry_iter | The handle for the entry iterator object to be examined. |

**Returns**

This function returns const char * with one of the following values:

| Value | Description |
|-------|-------------|
| NULL | If the specified iterator is NULL or does not reference a valid entry or the entry does not have any associated values. |
| value | Otherwise. |

# am_map_entry_iter_get_key()

Returns the key of the element currently referenced by the specified 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:

| Parameter | Description |
|---|---|
| entry_iter | The handle for the entry iterator object to be examined. |

### Returns

This function returns const char * with one of the following values:

| Value | Description |
|---|---|
| NULL | If the specified iterator is NULL or does not reference a valid entry. |
| key | Otherwise |

# am_map_entry_iter_get_values()

Returns an iterator object that can be used to enumerate all of the values associated with the entry referenced by the iterator you specify.

### Syntax

```
#include "am_map.h"
AM_EXPORT am_status_t
am_map_entry_iter_get_values(am_map_entry_iter_t entry_iter,
```

**Parameters**

This function takes the following parameters:

| Parameter | Description |
|---|---|
| entry_iter | The handle for the entry iterator object to be examined. |
| value_iter_ptr | Pointer to where the handle for the new value iterator object should be stored. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If no error was detected. |
| AM_NO_MEMORY | If unable to allocate memory for the value iterator object. |
| AM_INVALID_ARGUMENT | If the value_iter_ptr argument is NULL. |
| AM_NOT_FOUND | If the specified iterator is NULL or does not reference a valid entry. |

# am_map_entry_iter_is_entry_valid()

Determines if the specified iterator references a valid entry.

**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 parameters:

| Parameter | Description |
|---|---|
| entry_iter | The handle for the entry iterator object to be examined. |

**Returns**

This function returns boolean_t with one of the following values:

| Value | Description |
|---|---|
| 0 | If the specified iterator is NULL or does not reference a valid entry. |

| | |
|---|---|
| !0 | Otherwise. |

# am_map_entry_iter_next()

Advances the specified iterator to the next entry in the map specified when the iterator was 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 parameters:

| Parameter | Description |
|---|---|
| entry_iter | The handle for the entry iterator object to be modified. |

### Returns

This function returns boolean_t with one of the following values:

| Value | Description |
|---|---|
| 0 | If the specified iterator is NULL or does not reference a valid entry after being updated. |
| !0 | Otherwise. |

# am_map_erase()

Erases the specified key from the specified map.

### 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:

| Parameter | Description |
|---|---|
| map | The handle for the map object to be modified. |
| key | The key for the entry to erase. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If the entry was successfully erased from the map. |
| AM_INVALID_ARGUMENT | If either the map or key argument is NULL. |
| AM_NOT_FOUND | If the specified key is not currently in the map. |

# am_map_find()

Returns an iterator object that can be used to enumerate all of the values associated with the specified key.

**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:

| Parameter | Description |
|---|---|
| map | The handle for the map object to be examined. |
| key | The key for the entry to look up. |
| value_iter_ptr | Pointer to where the handle for the new value iterator object should be stored. |

**Returns**

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

| Value | Description |
|---|---|
| AM_SUCCESS | If no error was detected. |
| AM_NO_MEMORY | If unable to allocate memory for the value iterator object. |
| AM_INVALID_ARGUMENT | If the value_iter_ptr argument is NULL. |
| AM_NOT_FOUND | If the specified key could not be found in the map. |

**Details**

If the `value_iter_ptr` argument is non-NULL, then the location that it refers to will be set to NULL if an error is returned.

# am_map_find_first_value()

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

**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:

| Parameter | Description |
|---|---|
| map | The handle for the map object to be examined. |
| key | The key for the entry to look up. |

**Returns**

This function returns `const char *` with one of the following values:

| Value | Description |
|---|---|
| NULL | If the specified key could not be found in the map or the specified key had no associated values. |

| | |
|---|---|
| value | Otherwise, the first value associated with the key. |

# am_map_get_entries()

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

**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:

| Parameter | Description |
|---|---|
| map | The handle for the map object to be examined. |
| entry_iter_ptr | Pointer to where the handle for the new entry iterator object should be stored. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If no error was detected. |
| AM_NO_MEMORY | If unable to allocate memory for the entry iterator object. |
| AM_INVALID_ARGUMENT | If the entry_iter_ptr argument is NULL. |
| AM_NOT_FOUND | If the specified map contains no keys. |

**Details**

If the entry_iter_ptr argument is non-NULL, then the location that it refers to will be set to NULL if an error is returned.

# am_map_insert()

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

### 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:

| Parameter | Description |
|-----------|-------------|
| map | The handle for the map object to be modified. |
| key | The key for the entry. |
| value | The (new) value to be associated with the key. |
| replace | If non-zero, then the specified value replaces all of the existing values. Otherwise the specified value is added to the list of values associated with the specified key. |

### Returns

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

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

### Details

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

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

# am_map_size()

Returns the number of elements in the map.

### Syntax

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

### Parameters

This function takes the following parameters:

| Parameter | Description |
|-----------|-------------|
| map_ptr | The pointer to the map for which size is requested. |

### Returns

This function returns $size\_t$ with the size whose type is `size_t`.

# am_map_entry_iter_destroy()

Destroys the entry iterator object referenced by the provided handle.

### 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 parameters:

| Parameter | Description |
|-----------|-------------|
| entry_iter | The handle for the key iterator object to be destroyed. The handle may be NULL. |

### Returns

None

# am_map_value_iter_get()

Returns the value currently referenced by the specified 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 parameters:

| Parameter | Description |
|---|---|
| value_iter | The handle for the value iterator object to be examined. |

**Returns**

This function returns const char * with one of the following values:

| Value | Description |
|---|---|
| NULL | If the specified iterator is NULL or does not reference a valid value. |
| value | Otherwise |

# am_map_value_iter_is_value_valid()

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

**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 parameters:

| Parameter | Description |
|-----------|-------------|
| value_iter | The handle for the value iterator object to be modified. |

**Returns**

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

| Value | Description |
|-------|-------------|
| 0 | If the specified iterator is NULL or does not reference a valid value after being updated. |
| !0 | Otherwise |

am_map_value_iter_is_value_valid()

# Policy Functions

This chapter provides a reference to the public functions for using Sun Java™ System Access Manager 6 2005Q1 Access Management SDK policy objects. Function summaries include a short description, syntax, parameters and returns.

The following functions are contained in the header file `am_policy.h`.

- `am_policy_compare_urls()` on page 80
- `am_policy_destroy()` on page 80
- `am_policy_evaluate()` on page 81
- `am_policy_get_url_resource_root()` on page 82
- `am_policy_init()` on page 83
- `am_policy_is_notification_enabled()` on page 83
- `am_policy_notify()` on page 84
- `am_policy_resource_canonicalize()` on page 85
- `am_policy_resource_has_patterns()` on page 85
- `am_policy_result_destroy()` on page 86
- `am_policy_service_init()` on page 87

# am_policy_compare_urls()

Takes two url resources compares theme, and returns an 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
If the usePatterns is AM_TRUE, this function will consider occurrences of '*' in the policy resource name as wildcards. If usePatterns is AM_FALSE, '*' occurrences are taken as a literal characters.

### Returns
This function returns am_resource_match_t with one of the following values:

| Value | Description |
|---|---|
| EXACT_MATCH | If both the resource names exactly matched. |
| SUB_RESOURCE_MATCH | If the resourceName is a sub-resource to the resource name defined in the policy. |
| SUPER_RESOURCE_MATCH | If the resourcName is a ancestor of the policy resource name. |
| NO_MATCH | If the there is no kind of match between the policy resource and the requested resource name. |
| EXACT_PATTERN_MATCH | This result will be returned only if the policy is matches resource name. Distinction is not made whether it was a EXACT_MATCH or a pattern match. |

### Details
In cases of SUB/SUPER_RESOURCE_MATCH, if the usePatterns is * AM_TRUE, the patterns are sub/super matching patterns.

# am_policy_destroy()

Frees an initialized policy evaluator.

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

**Parameters**
This function takes the following parameters:

| Parameter | Description |
| --- | --- |
| policy | Opaque handle to the policy service to destroy. |

**Returns**
This function returns am_status_t with one of the following values:

| Value | Description |
| --- | --- |
| 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 resource and returns the policy result.

**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:

| Parameter | Description |
|---|---|
| policy_handle | Opaque handle to the policy service created by policy_service_init. |
| sso_token | User's SSO token to be used for evaluation. |
| resource_name | Name of resource to evaluate. |
| action_name | User's access action, such as GET or POST. |
| env_parameter_map | Any environment variables to be used for evaluation. |
| policy_response_map_ptr | Map to store user attributes from the policy evaluation call. |
| policy_result | Evaluation results. |

**Returns**

This function returns am_status_t with one of the following values:

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

# am_policy_get_url_resource_root()

Populates the pointer resourceRoot with the resource root.

**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 a URL resource name.

**Returns**

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

| Value | Description |
| --- | --- |
| AM_TRUE | Successful root extraction. |
| AM_FALSE | Otherwise |

# am_policy_init()

Initializes the policy evaluation engine.

### 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 parameters:

| Parameter | Description |
| --- | --- |
| properties | The properties to initialize the policy service with. |

### Returns

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

| Value | Description |
| --- | --- |
| 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 if notification is enabled in the SDK.

**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 parameters:

| Parameter | Description |
|---|---|
| policy_handle | The opaque policy service handle created from am_policy_service_init(). |

**Returns**
This function returns boolean_t with one of the following values:

| Value | Description |
|---|---|
| 0 | If notification is disabled. |
| non-zero | If notification is enabled. |

# am_policy_notify()

Refreshes 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:

| Parameter | Description |
|---|---|
| policy_handle | Opaque handle to the policy service |
| notification_data | The notification message as an XML String. |

| | |
|---|---|
| notification_data_len | Length of the notification data. |

### Returns
This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| 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()

Canonicalize the given resource name.

### 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:

| Parameter | Description |
|---|---|
| resource | Name of resource to be canonicalized |
| c_resource | Pointer to location where the canonicalized string will be placed. |
| | the value returned should be freed using free(). |

### Returns

None

# am_policy_resource_has_patterns()

Returns whether the given resource name has patterns such as '*'.

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

**Parameters**
This function takes the following parameters:

| Parameter | Description |
|---|---|
| resource_name | Name of the resource. |

**Returns**
This function returns `boolean_t` with one of the following values:

| Value | Description |
|---|---|
| true | If the resource has patterns. |
| false | Otherwise. |

# am_policy_result_destroy()

Destroys `am_policy_result` internal structures.

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

**Parameters**
This function takes the following parameters:

| Parameter | Description |
|---|---|
| result | The policy result to be destroyed. |

**Returns**
None

# am_policy_service_init()

Initializes one specific instance of 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:

| Parameter | Description |
|-----------|-------------|
| service_name | A name for the policy service. |
| instance_name | A name for the policy service instance. |
| rsrc_traits | Resource traits - see description of am_resource_traits_t in the structure section for more information. |
| service_config_properties | The properties to initialize the policy service with. |
| policy_handle_ptr | Handle to the policy service created. |

### Returns

This function returns am_status_t with one of the following values:

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

am_policy_service_init()

# Properties Functions

This chapter provides a reference to the properties map used by clients of the Sun Java™ System Access Manager 6 2005Q1 Remote Client SDK. Function summaries include a short description, syntax, parameters and returns.

The following functions are contained in the header file `am_properties.h`:

- am_properties_set() on page 102

- am_properties_store() on page 102

# am_properties_copy()

Makes a copy of a properties object.

**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:

| Parameter | Description |
|---|---|
| source_properties | The handle for the properties object to be copied. |
| properties_ptr | A pointer to where to store the handle of the new created properties object. |

**Returns**
This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If a 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. |

# 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:

| Parameter | Description |
|---|---|
| properties_ptr | A pointer to where to store the handle of the new created properties object. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| 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. |

# am_properties_destroy()

Destroys the properties object referenced by the provided handle.

**Syntax**

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

**Parameters**

This function takes the following parameters:

| Parameter | Description |
|---|---|
| properties | The handle for the properties object to be destroyed. |

**Returns**

None

# am_properties_get()

This function and all functions beginning with am_properties_get retrieve values from the properties map. The following parameters and exceptions are common to all functions in the am_propteries_get collection. Additional return values may be specified some functions.

### 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:

| Parameter | Description |
|-----------|-------------|
| properties | Handle to the properties object to be examined. |
| key | Name of the property to look up. |
| value_ptr | A pointer to where to store the value associated with the default value. |
| default_value | Default value to use if there is no value associated with the specified key. |

### Returns

This function returns the unparsed string form of the value associated with one of the following keys:

| Value | Description |
|-------|-------------|
| 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. |

# am_properties_get_boolean()

Retrieves values from the properties map.

### 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

See am_properties_get().

### Returns

- The unparsed string form of the value associated with the specified key. See am_properties_get().

- A value stored in value_ptr with one of the following values:

| Value | Description |
| --- | --- |
| !0 | If the value associated with the specified key is one of: true, on, or yes. |
| 0 | If the value associated with the specified key is one of: false, off, or no. |

### Details

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

See also am_properties_get().

# am_properties_get_boolean_with_default()

Retrieves values from the properties map.

**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**

See am_properties_get().

**Returns**

- The unparsed string form of the value associated with the specified key. See am_properties_get().

- A value stored in value_ptr with one of the following values:

| Value | Description |
|-------|-------------|
| !0 | If the value associated with the specified key is one of: true, on, or yes. |
| 0 | If the value associated with the specified key is one of: false, off, or no. |

**Details**

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

See also am_properties_get().

# am_properties_get_entries()

Returns an iterator object that can be used to enumerate all of the entries in the specified properties object. See also am_properties_get().

**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:

| Parameter | Description |
|---|---|
| properties | The handle for the properties object to be examined |
| properties_iter_ptr | Pointer to where the handle for the new properties iterator object should be stored. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If no error was detected. |
| AM_NO_MEMORY | If unable to allocate memory for the properties iterator object. |
| AM_INVALID_ARGUMENT | If the properties_iter_ptr argument is NULL. |
| AM_NOT_FOUND | If the specified properties object contains no entries. |

**Details**

If the properties_iter_ptr argument is non-NULL, then the location that it refers to will be set to NULL if an error is returned.

See also am_properties_get().

# am_properties_get_signed()

Retrieves values from the properties map.

**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**
See am_properties_get().

**Returns**
This function returns the value stored in `value_ptr` which is the signed integer value associated with the specified key.

**Details**
If the associated value cannot be parsed as an integer or cannot be represented in the range `LONG_MIN` to `LONG_MAX`, then `AM_INVALID_VALUE` will be returned.

See also am_properties_get().

# am_properties_get_signed_with_default()

Retrieve values from the properties map.

**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**
See am_properties_get().

**Returns**
This function returns the value stored in `value_ptr` which is the signed integer value associated with the specified key.

**Details**
If the associated value cannot be parsed as an integer or cannot be represented in the range `LONG_MIN` to `LONG_MAX`, then `AM_INVALID_VALUE` will be returned.

See also am_properties_get().

# am_properties_get_unsigned()

See am_properties_get().

**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**
See am_properties_get().

**Returns**
This function returns the unsigned integer value associated with the specified keyDetails.

**Details**
See am_properties_get().

# am_properties_get_unsigned_with_default()

See am_properties_get().

**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**
See am_properties_get().

**Returns**
This function returns the unsigned integer value associated with the specified keyDetails.

**Details**

See am_properties_get().

# am_properties_get_with_default()

Retrieves values from the properties map.

### 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

See am_properties_get().

### Returns

See am_properties_get().

### Details

See am_properties_get().

# am_properties_is_set()

Determines whether the object contains property with the specified name.

### 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:

| Parameter | Description |
| --- | --- |

| properties | Handle to the properties object to be examined. |
| --- | --- |
| key | Name of the property to look up. |

**Returns**

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

| Value | Description |
| --- | --- |
| !0 | If the property has a value. |
| 0 | Otherwise |

# am_properties_iter_destroy()

Destroys the properties iterator object referenced by the provided handle.

**Syntax**

```
#include "am_properties.h"
AM_EXPORT void
am_properties_iter_destroy(am_properties_iter_t properties_iter);
```

**Parameters**

This function takes the following parameters:

| Parameter | Description |
| --- | --- |
| properties_iter | The handle for the key iterator object to be destroyed. The handle may be NULL. |

**Returns**

None

# am_properties_iter_get_key()

Returns the key of the element currently referenced by the specified iterator.

**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 parameters:

| Parameter | Description |
| --- | --- |
| properties_iter | The handle for the properties iterator object to be examined. |

**Returns**

This function returns const char * with one of the following values:

| Value | Description |
| --- | --- |
| NULL | If the specified iterator is NULL or does not reference a valid entry. |
| key | Otherwise. |

# am_properties_iter_get_value()

Returns the value of the element currently referenced by the specified iterator.

**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:

| Parameter | Description |
| --- | --- |
| properties_iter | The handle for the properties iterator object to be examined. |

**Returns**

This function returns const char * with one of the following values:

| Value | Description |
| --- | --- |

| NULL | If the specified iterator is NULL or does not reference a valid entry. |
|------|------|
| value | Otherwise. |

# am_properties_load()

Loads property information from the specified 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:

| Parameter | Description |
|-----------|-------------|
| properties | Handle to the properties object to be modified. |
| file_name | Name of the file from which to load the property information. |

**Returns**
This function returns am_status_t with one of the following values:

| Value | Description |
|-------|-------------|
| AM_SUCCESS | If no error is detected. |
| 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. |

**Details**
The file is expected to use the standard Java Properties file syntax.

# am_properties_set()

Sets the value associated with the specified key.

### 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:

| Parameter | Description |
|-----------|-------------|
| properties | Handle to the properties object to be modified. |
| key | The key to modify. |
| value | The value to associate with the specified key. |

### Returns

This function returns am_status_t with one of the following values:

| Value | Description |
|-------|-------------|
| 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. |

### Details

The specified value will replace any previously existing value.

# am_properties_store()

Stores the property information 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:

| Parameter | Description |
|---|---|
| properties | Handle to the properties object to be stored. |
| file_name | Name of the file in which to store the property information. |

**Returns**
This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If no error is detected. |
| AM_NSPR_ERROR | If there is a problem writing the properties to the file. |
| AM_INVALID_ARGUMENT | If properties or file_name is NULL or file_name points to an empty string. |

am_properties_store()

# Single Sign-On Functions

This chapter provides a reference to the public functions you can use to implement Single Sign-on (SSO) in Sun Java™ System Access Manager 6 2005Q. Function summaries include a short description, syntax, parameters and returns.

The following functions are contained in the header file am_sso.h.

# am_sso_add_listener()

Adds a listener for the any SSO token's change events.

### 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:

| Parameter | Description |
| --- | --- |
| listener | The token change event listener. |
| args | Arguments to pass to the listener. |
| dispatch_to_sep_thread | Call the listener in a separate thread from an internal thread pool. This 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 am_status_t with one of the following values:

| Value | Description |
| --- | --- |
| AM_SUCCESS | If the listener was successfully added. |

| AM_INVALID_ARGUMENT | If `sso_token_handle` or listener is invalid, or if `notification_url` is not set and no notification url is provided in the properties file. |
|---|---|
| AM_NOTIF_NOT_ENABLED | If notification is not enabled and the `notification_url` input parameter is invalid. |
| AM_FAILURE | If any other error occurred. |

**Details**

Caller must either provide a URL to this function or have notification enabled with a valid notification URL in the properties file used to initialize SSO in `am_sso_init()`. The URL must point to a HTTP host and port that listens for notification messages from the server.

Notification messages are in XML. XML Notification messages received from the server should be passed to as a string (`const char *`) to `am_notify()`, which will parse the message and invoke listeners accordingly.

See the C API samples for more information.

When the listener is called, the `sso_token_handle` that is passed to the listener is a temporary one containing the updated session information from the server. Note that it is not the original `sso_token_handle` passed to `am_sso_add_sso_token_listener()`.

Once added the listener will be called for any and all session event change notification. It will not be removed after it is called once like `am_sso_add_sso_token_listener`.

# am_sso_add_sso_token_listener()

Adds an SSO token listener for the SSO token's change events.

**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:

| Parameter | Description |
| --- | --- |
| sso_token_handle | The session handle containing the SSO token id to listen for. The handle will be filled with the session information from the notification message. Any existing contents will be overwritten. |
| listener | The token change event listener. |
| args | Arguments to pass to the listener. |
| dispatch_to_sep_thread | Calls the listener in a separate thread from an internal thread pool. This allows am_notify to return immediately upon parsing the notification message rather than waiting for the listener function(s) to finish before returning. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
| --- | --- |
| AM_SUCCESS | If the listener was successfully added. |
| AM_INVALID_ARGUMENT | If sso_token_handle or listener is invalid, or if notification_url is not set and no notification URL is provided in the properties file. |
| AM_NOTIF_NOT_ENABLED | If notification is not enabled and the notification_url input parameter is invalid. |
| AM_FAILURE | If any other error occurred. |

**Details**

Caller must either provide a URL to this function or have notification enabled with a valid notification URL in the properties file used to initialize SSO in am_sso_init(). The URL must point to a HTTP host and port that listens for notification messages from the server.

Notification messages are in XML. XML Notification messages received from the server should be passed to as a string (const char *) to am_notify(), which will parse the message and invoke listeners accordingly.

See the C API samples for more information.

When the listener is called, the `sso_token_handle` that is passed to the listener is a temporary one containing the updated session information from the server. Note that it is not the original `sso_token_handle` passed to `am_sso_add_sso_token_listener()`.

Once a listener has been called it is removed from memory; a listener is called only once.

# am_sso_create_sso_token_handle()

Creates a handle to session information.

### 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:

| Parameter | Description |
|---|---|
| sso_token_handle | Pointer to SSO token handle which will be assigned an handle if the session validation is successful. |
| sso_token_id | String representation session identifier. |
| reset_idle_timer | When querying for session information. |

### Returns

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

| Value | Description |
|---|---|
| AM_SUCCESS | If session validation was successful and a handle was successfully created. |
| AM_SERVICE_NOT_INITIALIZED | If SSO token service was not initialized. SSO token service must be initialized by calling `am_sso_init()` any call to `am_sso_*` can be made. |
| 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 handle to session information.

### 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 parameters:

| Parameter | Description |
|---|---|
| sso_token_handle | SSO token handle to be de-allocated. |

### Returns

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

| Value | Description |
|---|---|
| AM_SUCCESS | If the memory release process was successful. |
| AM_INVALID_ARGUMENT | If the session_token_handle parameter is NULL. |
| AM_FAILURE | If any other error occurred. |

### Details

This function does NOT log out the user or invalidate the session.

# am_sso_get_auth_level()

Gets the auth level for this session.

### 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 parameters:

| Parameter | Description |
|---|---|
| sso_token_handle | The SSO token handle. |

**Returns**

This function returns the auth level of this session handle; returns ULONG_MAX if there was any error.

# am_sso_get_auth_type()

Gets the auth type for this session.

### 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 parameters:

| Parameter | Description |
|---|---|
| sso_token_handle | The SSO token handle. |

**Returns**

This function returns the auth type of this session handle. NULL if there was any error.

# am_sso_get_host()

Gets the host address for this session.

**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 parameters:

| Parameter | Description |
| --- | --- |
| sso_token_handle | The SSO token handle. |

**Returns**
This function returns the host name of this session handle as given by the "Host" property. NULL if the "Host" property is not set or does not have a value.

# am_sso_get_idle_time

Gets idle time associated with this session 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 parameters:

| Parameter | Description |
| --- | --- |
| sso_token_handle | The SSO token handle. |

**Returns**
This function returns the idle time of the session handle in seconds.

(time_t) -1 if token is invalid or some error occurs. Detailed error is logged.

# am_sso_get_max_idle_time()

Gets the max idle time for this session.

### 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:

| Parameter | Description |
|---|---|
| sso_token_handle | The SSO token handle. |

### Returns

This function returns the max idle time for this session handle in seconds. (time_t) -1 if there was any error.

# am_sso_get_max_session_time()

Gets the max session time for this session.

### Syntax

```
#include "am_sso.h"
```

### Parameters

This function takes the following parameters:

| Parameter | Description |
|---|---|
| sso_token_handle | The SSO token handle. |

### Returns

This function returns the max session time of this session handle in seconds. (time_t) -1 if there was any error.

# am_sso_get_principal()

Gets the principal of this session.

### Syntax

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

This function takes the following parameters:

| Parameter | Description |
|-----------|-------------|
| sso_token_handle | The SSO token handle. |

### Returns

This function returns the principal of this session handle, NULL if the sso_token handle is invalid or any other error occurred.

# am_sso_get_principal_set()

Gets the set of principals of this session. A session can have more than one principal.

### 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 parameters:

| Parameter | Description |
|-----------|-------------|
| sso_token_handle | The SSO token handle. |

### Returns

This function returns the set of principals of this session handle, NULL if the principal property is not set or has no value.

# am_sso_get_property()

Gets the value of a session property.

### 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:

| Parameter | Description |
|---|---|
| sso_token_handle | The SSO token handle. |
| property_key | The name of property to get. |
| check_if_session_valid | Whether to check if session is valid first. If true and session is invalid, NULL will always be returned. |

### Returns

This function returns the value of the session property. NULL if property is not set or does not have a value.

# am_sso_get_sso_token_id()

Gets the SSO token ID for this session.

### 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:

| Parameter | Description |
|---|---|
| sso_token_handle | The SSO token handle. |

**Returns**

This function returns the SSO token ID of this session. NULL if sso_token_handle is invalid or any other error occurred.

# am_sso_get_time_left()

Gets the time left of this session 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:

| Parameter | Description |
|---|---|
| sso_token_handle | The SSO token handle. |

### Returns

This function returns the time left of this session handle in seconds. (time_t) -1 if token is invalid or some error occurs.

### Details

Detailed error is logged.

# am_sso_init()

Initializes the SSO module in the C API.

### Syntax

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

### Parameters

This function takes the following parameters:

| Parameter | Description |
|---|---|

| | |
|---|---|
| property_map | Properties object to initialize sso with. |

**Returns**

This function returns am_status_t with one of the following values:

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

**Details**

This call must be made before any calls to am_sso_* functions.

# am_sso_invalidate_token()

Invalidates or destroys the session on the server.

**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 parameters:

| Parameter | Description |
|---|---|
| sso_token_handle | SSO token handle of session to be invalidated. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If session was successfully invalidated. |
| AM_INVALID_ARGUMENT | If the sso_token_handle is invalid. |
| AM_SERVICE_NOT_INITIALIZED | If the sso token service was not initialized with am_sso_init(). |
| AM_SERVICE_NOT_AVAILABLE | If server returned service not available. |

| | |
|---|---|
| AM_HTTP_ERROR | If HTTP error encountered while communicating with server. |
| AM_ERROR_PARSING_XML | If error parsing XML from server. |
| AM_ACCESS_DENIED | If access denied while communicating with server. |
| AM_FAILURE | If any other error occurred. |

**Details**

If successful the session handler in input argument will have state invalid after this call.

Note: Does not free the sso_token_handle input parameter. Call am_sso_destroy_sso_token_handle() to free memory for the handle itself.

# am_sso_is_valid_token()

Checks if a token is valid.

**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 parameters:

| Parameter | Description |
|---|---|
| sso_token_handle | SSO token to check if valid. |

**Returns**

This function returns boolean_t with one of the following values:

| Value | Description |
|---|---|
| B_TRUE | If SSO token is valid. |
| B_FALSE | If SSO token is invalid or any other error occurred. |

**Details**

This call looks in the passed sso_token_handle to check for validity; it does *not* go to the server.

# am_sso_refresh_token()

Refreshes an SSO token 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 parameters:

| Parameter | Description |
|---|---|
| sso_token_handle | SSO token to refresh. |

### Returns

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If SSO token could be refreshed with no errors. |
| AM_INVALID_ARGUMENT | If the input parameter is invalid. |
| AM_SERVICE_NOT_INITIALIZED | If SSO token service is not initialized. SSO token service must be initialized by calling am_sso_init() before any call to am_sso*. |
| AM_SERVICE_NOT_AVAILABLE | If server returned service not available. |
| AM_HTTP_ERROR | If HTTP error encountered while communicating with server. |
| AM_ERROR_PARSING_XML | If error parsing XML from server. |
| AM_ACCESS_DENIED | If access denied while communicating with server. |
| AM_SESSION_FAILURE | If the session validation failed. |
| AM_FAILURE | If any other error occurred. |

**Details**

This goes to the server to get latest session info and update it in the `sso_token_handle` input parameter like `am_sso_validate_token()`. However it also refreshes the last access time of the session.

# am_sso_remove_listener()

Removes an SSO token listener for any SSO token's change events.

### 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 parameters:

| Parameter | Description |
|-----------|-------------|
| listener | The change event listener. |

### Returns

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

| Value | Description |
|-------|-------------|
| AM_SUCCESS | If the listener was successfully removed. |
| AM_INVALID_ARGUMENT | If listener was NULL. |
| AM_NOT_FOUND | If listener was not found. |
| AM_FAILURE | If any other error occurred. |

### Details

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

# am_sso_remove_sso_token_listener()

Removes an SSO token listener for the SSO token's change events.

**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:

| Parameter | Description |
| --- | --- |
| sso_token_handle | The session handle containing the sso token id for the listener. |
| listener | The token change event listener. |

**Returns**
This function returns am_status_t with one of the following values:

| Value | Description |
| --- | --- |
| AM_SUCCESS | If the listener was successfully removed. |
| AM_INVALID_ARGUMENT | If sso_token_id or listener is invalid or NULL. |
| AM_NOT_FOUND | If listener was not found for the sso token id. |
| AM_FAILURE | If any other error occurred. |

**Details**
If am_sso_token_add_listener() was called more than once with the same listener function, all instances of the listener function will be removed.

# am_sso_set_property()

Sets a property in the session.

**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:

| Parameter | Description |
|---|---|
| sso_token_handle | The session handle. |
| name | The property name. |
| value | The property value. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| 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. |

**Details**

Session handle for this token ID obtained before this call will not be current (not have the newly set property) after this call. Call am_sso_validate_token() to update the handle with the new set of properties.

# am_sso_validate_token()

Validates an SSO token.

**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 parameters:

| Parameter | Description |
|---|---|
| sso_token_handle | SSO token to validate. |

**Returns**

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

| Value | Description |
|---|---|
| AM_SUCCESS | If SSO token is valid, session handle is updated. |
| AM_INVALID_SESSION | If the session is invalid, session handle is updated. |
| AM_INVALID_ARGUMENT | If the input parameter is invalid. |
| AM_SERVICE_NOT_INITIALIZED | If SSO token service is not initialized. sso token service must be initialized by calling am_sso_init() before any call to am_sso*. |
| AM_SERVICE_NOT_AVAILABLE | If server returned service not available. |
| AM_HTTP_ERROR | If HTTP error encountered while communicating with server. |
| AM_ERROR_PARSING_XML | If error parsing XML from server. |
| AM_ACCESS_DENIED | If access denied while communicating with server. |
| AM_FAILURE | If any other error occurred. |

**Details**

This call will go to the server to get the latest session info and update the `sso_token_handle` input parameter. The `sso_token_handle` input parameter 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.

am_sso_validate_token()

# Web Functions

This chapter provides a reference to the functions in the C SDK intended for use by only web agents of Sun Java™ System Access Manager 6 2005Q1.  Function summaries include a short description, syntax, parameters and returns.

The following functions are contained in the header file am_web.h:

# am_web_clean_post_urls()

Cleans up data structure containing dummy post url, action url and unique key.

### 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 parameters:

| Parameter | Description |
| --- | --- |
| posturl_struct | Pointer to POST preservation URL data structure post_urls_t. |

**Returns**

None

# am_web_cleanup()

Cleans up any memory called by the am_web_* functions.

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 am_status_t with one of the following values:

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

# am_web_create_post_page()

Creates the HTMLform with the javascript that submits the POST with the 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:

| Parameter | Description |
|-----------|-------------|
| key | Unique key to identify POST data entry. It is used to remove post data once the page is re-posted. |
| postdata | POST data entry as a browser encoded string actionurl. |
| actionurl | POST destination URL. |

**Returns**
This function returns char * with one of the following values:

| Value | Description |
|-------|-------------|
| char * | POST form to be resubmitted. |

# am_web_create_post_preserve_urls()

Constructs dummy post url, action url and unique key.

**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 parameters:

| Parameter | Description |
|-----------|-------------|
| request_url | The request URL for POST in the HTTP request. |

**Returns**

This function returns `post_urls_t *` with one of the following values:

| Value | Description |
|-------|-------------|
| `post_urls_t` | Data structure that contains Dummy redirect URL, POST destination URL and POST preservation key. |

**Details**

Dummy redirect URL is filtered by web server SAF to identify POST preservation redirects from general redirects. All three of these variables are required for POST preservation.

# 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 parameters:

| Parameter | Description |
|-----------|-------------|
| `memory` | Memory allocated by a am_web_* routine to be freed. |

**Returns**

None

# am_web_get_agent_server_host()

Retrieves the name of the Agent Server Host.

**Syntax**
```
#include "am_web.h"
AM_WEB_EXPORT const char *
am_web_get_agent_server_host();
```

**Parameters**
This function does not take any parameters.

**Returns**
This function returns const char * with one of the following values:

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

# am_web_get_agent_server_port()

Retrieves the name of the Agent Server Port.

**Syntax**
```
#include "am_web.h"
AM_WEB_EXPORT int
am_web_get_agent_server_port();
```

**Parameters**
This function does not take any parameters.

**Returns**
This function returns  int with one of the following values:

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

# 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 does not take any parameters.

**Returns**
This function returns const char * with one of the following values:

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

# am_web_get_notification_url()

Retrieves the name of the Access Manager notification URL.

**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 const char * with one of the following values:

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

# am_web_get_parameter_value()

Gets the value of the given query parameter from the given 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:

| Parameter | Description |
|---|---|
| inpQuery | URL to look for the query parameter. |
| param_name | Name of the query parameter. |
| param_value | Pointer to be filled with the value of the param_name query parameter in the given URL if found. |
| | The returned paramter value should be freed by the caller using am_web_free(). |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If the query parameter was found in the URL. |
| AM_INVALID_ARGUMENT | If any of the arguments is NULL. |
| AM_NOT_FOUND | If the query paramter is not found. |
| AM_NO_MEMORY | If memory could not be allocated for the query parameter value. |
| AM_FAILURE | If any other error occurred. |

# am_web_get_redirect_url()

Returns a string representing the Access Manager URL that web agent should redirect to. For example, if the status is AM_INVALID_SESSION and CDSSO is not enabled, the redirect URL would be the Access Manager login URL as configured in the AMAgent.properties file and associated query parameters.

**Syntax**
```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_get_redirect_url(am_status_t status,
                        const am_map_t advice_map,
                        const char *goto_url,
                        const char* function,
                        char ** redirect_url);
```

**Parameters**
This function takes the following parameters:

| Parameter | Description |
| --- | --- |
| status | The status from am_web_is_access_allowed. |
| advice_map | Any advice map from policy evaluation results. |
| goto_url | Original URL accessed by the user, for IS to redirect user to after successful authentication with the Access Manager. |
| redirect_url | A pointer to contain the resulting Access Manager redirect URL. |

**Returns**
This function returns am_status_t with one of the following values:

| Value | Description |
| --- | --- |
| AM_SUCCESS | if the call was successful. |
| AM_* | If any error occurs, the type of error indicated by the status value. |

**Details**
The string may either redirect the user to the login URL or the access denied URL. If the redirection is to the login URL then the URL will include any existing information specified in the URL from the configuration file, like org value etc., followed by the specified goto parameter value, which will be used by Access Manager after the user has successfully authenticated.

If the redirect_url returned is NOT NULL, the caller of this function must call am_web_free_memory(void *) to free the pointer.

# am_web_get_token_from_assertion()

Returns the SSO Token from the given 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:

| Parameter | Description |
|---|---|
| assertion | The SAML assertion as an XML string. |
| token | Pointer to contain the SSO Token ID. |
| | The returned SSO Token ID should be freed using am_web_free(). |

### Returns

This function returns am_status_t with one of the following values:

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

# am_web_handle_notification()

Handles notification data received by an agent.

### 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:

| Parameter | Description |
|---|---|
| data | The notification message as an XML string. |

| data_length | Length of the notification message. |
| --- | --- |

**Returns**

None

**Details**

This code handles generating logging messages for the event and any error that may occur during the processing of the notification.

# am_web_http_decode()

URL decodes the given 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:

| Parameter | Description |
| --- | --- |
| string | the URL encoded string |
| len | Length of the string. |

**Returns**

This function returns the URL decoded value of the URL encoded string, or NULL if any error occurred.

The returned value should be freed by calling am_web_free().

# am_web_init()

Initializes the Agent Toolkit.

### Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_init(const char *config_file);
```

### Parameters

This function takes the following parameters:

| Parameter | Description |
|---|---|
| config_file | Path to the agent configuration file, for example, /etc/opt/AMAgent.properties. |

### Returns

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| 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.

### 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:

| Parameter | Description |
|---|---|
| sso_token | The sso_token from the Access Manager cookie. This parameter may be NULL if there is no cookie present. |
| url | The URL whose accessibility is being determined. This parameter may not be NULL. |

| | |
|---|---|
| action_name | The action (GET, POST, etc.) being performed on the specified URL. This parameter may not be NULL. |
| client_ip | The IP address of the client attempting to access the specified URL. If client IP validation is turned on, then this parameter may not be NULL. |
| env_parameter_map | A map containing additional information about the user attempting to access the specified URL. This parameter ay not be NULL. |
| advices_map_ptr | An output parameter where an am_map_t can be stored if the policy evaluation produces any advice information. This parameter may not be NULL. |

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If the evaluation was performed successfully and access is to be allowed to the specified resource. |
| 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 sso_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 whether CDSSO 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 true if CDSSO is enabled and false otherwise.

# am_web_is_debug_on()

Returns debug is on, that is, if 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 boolean_t with one of the following values:

| Value | Description |
|-------|-------------|
| true | If the log level is set to anything greater than 0. |
| false | Otherwise. |

# am_web_is_in_not_enforced_ip_list()

Returns true if the given IP address is present in the list of not enforced IP addresses.

### 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 parameters:

| Parameter | Description |
|-----------|-------------|
| ip | The IP address. |

**Returns**

This function returns boolean_t with one of the following values:

| Value | Description |
| --- | --- |
| true | If the IP is in the not enforced IP address list. |
| false | Otherwise. |

# am_web_is_in_not_enforced_list()

Returns true if the URL being accessed by the user is in the not enforced list.

**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:

| Parameter | Description |
| --- | --- |
| url | The URL being accessed by the user |
| path_info | Path info of the URL. |

**Returns**

This function returns boolean_t with one of the following values:

| Value | Description |
| --- | --- |
| true | If the URL is in the not enforced list. |
| false | Otherwise. |

# am_web_is_max_debug_on()

Returns true if 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 boolean_t with one of the following values:

| Value | Description |
|-------|-------------|
| true | If the log level is set to 5. |
| false | Otherwise. |

# am_web_is_notification()

Returns true if the given URL is the notification 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:

| Parameter | Description |
|-----------|-------------|
| request_url | The request URL |

**Returns**

This function returns am_web_is_notification with one of the following values:

| Value | Description |
|-------|-------------|
| true | If the URL is the notification URL of the agent as set in AMAgent.properties. |
| false | Otherwise. |

# am_web_is_postpreserve_enabled()

Finds out if POST data preservation is enabled by clients through
`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 `boolean_t` with one of the following values:

| Value | Description |
| --- | --- |
| True | If POST preservation is switched on. |
| False | If POST preservation is switched off. |

# am_web_is_valid_fqdn_url()

Returns if the requested URL is a Valid FQDN resource, that is if the host is a fully
qualified domain name such as myhost.mydomain.com as configured in
AMAgent.properties.

### 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 `boolean_t` with one of the following values:

| Value | Description |
| --- | --- |
| true | If the URL is using a fully qualified domain name. |
| 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 parameters:

| Parameter | Description |
|-----------|-------------|
| fmt | Formatted string as in printf. |

### Returns

None

# am_web_log_auth()

Log the given access allowed or denied message to the Access Manager's 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:

| Parameter | Description |
|-----------|-------------|
| access_type | AM_ACCESS_ALLOW or AM_ACCESS_DENY. |
| message | Any message for the log. |

### Returns

This function returns boolean_t with one of the following values:

| Value | Description |
|-------|-------------|

| true | If the call was successful. |
| --- | --- |
| 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 parameters:

| Parameter | Description |
| --- | --- |
| fmt | A formatted string as in printf. |

### Returns

None

# 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 parameters:

| Parameter | Description |
| --- | --- |
| fmt | A formatted string as in printf to be logged. |

**Returns**

None

# 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 parameters:

| Parameter | Description |
|-----------|-------------|
| fmt | Formatted string like in printf to be logged. |

**Returns**

None

# am_web_log_max_debug()

Log the given message at max 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:

| Parameter | Description |
|-----------|-------------|
| fmt | Formatted string as in printf to be logged. |

**Returns**

None

# 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:

| Parameter | Description |
| --- | --- |
| fmt | A formatted string as in printf to be logged. |

### Returns

None

# am_web_postcache_data_cleanup()

Cleans up data structure containing post string value, redirect url.

### 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 parameters:

| Parameter | Description |
| --- | --- |
| const | am_web_postcache_data_t<br>Pointer to POST data entry |

**Returns**
None

# 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:

| Parameter | Description |
|-----------|-------------|
| key | POST data preservation key for every entry. |
| value | Structure to store POST data value and redirect URL. |

### Returns
This function returns boolean_t with one of the following values:

| Value | Description |
|-------|-------------|
| True | If insertion was successful. |
| False | If insertion was not successful. |

# am_web_postcache_lookup()

Looks up POST 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:

| Parameter | Description |
| --- | --- |
| key | Key to search POST data entry in POST data structure |

**Returns**

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

| Value | Description |
| --- | --- |
| am_web_postcache_data_t | Data structure containing POST data and redirect URL |

# am_web_postcache_remove()

Removes POST 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 parameters:

| Parameter | Description |
| --- | --- |
| key | Key to remove an entry from POST data structure. |

**Returns**

None

# am_web_remove_parameter_from_query()

Removes the given query parameter from the URL, if it is in the URL.

**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:

| Parameter | Description |
|-----------|-------------|
| inpString | The original URL |
| remove_str | The query parameter to be removed |
| outString | Pointer to location where a new URL with the query parameter removed will be inserted. |
| | The value returned should be freed using am_web_free(). |

**Returns**

This function returns am_status_t with one of the following values:

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

# Miscellaneous Functions

This chapter provides a reference to various Sun Java™ System Access Manager 6 2005Q1 functions that do not belong in other collections. Function summaries include a short description, syntax, parameters and returns, and header file.

The following functions are included in this chapter:

## am_cleanup()

Cleans up any memory allocated by C SDK.

This function must be called when a caller is done with C SDK interfaces to cleanup memory allocated by the C SDK.

### Syntax

```
#include "am_h.h"
AM_EXPORT am_status_t
am_cleanup(void);
```

**Parameters**

This function takes no parameters.

**Returns**

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

| Value | Description |
|---|---|
| AM_SUCCESS | If XML message was successfully parsed and processed. |
| AM_INVALID_ARGUMENT | If any input parameter is invalid. |
| AM_FAILURE | If any other error occurred. |

**Details**

This should be called only once at the end of C SDK calls, after which the initialize functions `am_*_init()` must be called again to initialize the C SDK before using any of its interfaces.

Any properties input parameter given to the init functions `am_sso_init()`, `am_auth_init()` or `am_policy_init()` should be destroyed only after `am_cleanup` is called.

# am_notify()

Parses and processes an SSO or policy notification message as an XML string. If the message is an SSO notification, any SSO Token listeners registered using `am_sso_add_listener()` will be called. If the notification 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 if the policy module in the C SDK has been initialized (using `am_policy_init()` and `am_policy_service_init()`).

**Syntax**

```
#include "am_notify.h"
AM_EXPORT am_status_t
am_notify(const char *xmlmsg, am_policy_t policy_handle);
```

**Parameters**

This function takes the following parameters:

| Parameter | Description |
|---|---|
| xmlmsg | XML message containing the notification message. |

| policy_handle_t | The policy handle created from am_policy_service_init(). NULL if policy is not initialized or not used. |
|---|---|

**Returns**

This function returns am_status_t with one of the following values:

| Value | Description |
|---|---|
| AM_SUCCESS | If XML message was successfully parsed and processed. |
| AM_INVALID_ARGUMENT | If any input parameter is invalid. |
| AM_ERROR_PARSING_XML | If there was an error parsing the XML message. |
| AM_ERROR_DISPATCH_LISTENER | If there was an error dispatching the listener(s). |
| AM_FAILURE | If any other error occurred. |

**Details**

This function should be called by the service listening on the notification URL given in the properties file if notification is enabled.

# am_string_set_allocate()

Allocates space for an am_string_set_t and space for size strings. Also initializes size to the given size.

**Syntax**

```
#include "am_string.h"
AM_EXPORT am_string_set_t *
am_string_set_allocate(int size);
```

**Parameters**

This function takes the following parameters:

| Parameter | Description |
|---|---|
| size | Size of set to allocate. |

**Returns**

This function returns a pointer to allocated am_string_set_t, or NULL if size is less than 0.

# am_string_set_destroy()

Frees memory held by the parameter, by freeing each string in the set of strings, followed by the strings pointer, followed by the struct 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 parameters:

| Parameter | Description |
| --- | --- |
| string_set | The am_string_set_t pointer to be freed. |

### Returns

None

# am_status_to_name()

Returns the name of the given status code as a string. For example, the name of `AM_SUCCESS` is `"AM_SUCCCESS"`.

### Syntax

```
#include "am_types.h"
AM_EXPORT const char *
am_status_to_name(am_status_t status);
```

### Parameters

This function takes the following parameters:

| Parameter | Description |
| --- | --- |
| status | The status code. |

### Returns

This function returns the name of the status code as a `const char *`.

# am_status_to_string()

Returns the message for the given status code. For example, the message for AM_SUCCESS is "success".

### Syntax

```
#include "am_types.h"
AM_EXPORT const char *
am_status_to_string(am_status_t status);
```

### Parameters

This function takes the following parameters:

| Parameter | Description |
|-----------|-------------|
| status | The status code. |

### Returns

This function returns the message for the status code as a const char *.

### Details

The header file for this function is am_types.h

# am_http_cookie_encode()

URL encodes a HTTP cookie.

### Syntax

```
#include "am_utls.h"
AM_EXPORT am_status_t
am_http_cookie_encode(const char *cookie, char *buf, int len);
```

### Parameters

This function takes the following parameters:

| Parameter | Description |
|-----------|-------------|
| cookie | The cookie to be URL encoded. |
| buf | The buffer to put the encoded cookie. |
| len | The size of the buffer. |

**Returns**

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

| Value | Description |
|---|---|
| AM_SUCCESS | If the cookie was successfully encoded and copied into buf. |
| AM_INVALID_ARGUMENT | If the cookie or buffer was NULL or len was smaller than the size of the encoded value. |
| AM_FAILURE | Other error occurred while encoding cookie. |

# am_http_cookie_decode()

URL decodes a HTTP cookie.

**Syntax**

```
#include "am_utls.h"
AM_EXPORT am_status_t
am_http_cookie_decode(const char *cookie, char *buf, int len);
```

**Parameters**

This function takes the following parameters:

| Parameter | Description |
|---|---|
| cookie | The cookie to be URL decoded. |
| buf | The buffer to put the decoded cookie |
| len | The size of the buffer |

**Returns**

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

| Value | Description |
|---|---|
| AM_SUCCESS | If the cookie was successfully decoded and copied into buf. |
| AM_INVALID_ARGUMENT | If the cookie or buffer was NULL or len was smaller than the size of the decoded value. |
| AM_FAILURE | Other error occurred while decoding cookie. |

# Glossary

For a list of terms that are used in this documentation set, refer to the *Sun Java™ Enterprise System Glossary* (`http://docs.sun.com/doc/816-6873`)