Sun Java System Access Manager 7 2005Q4 C API Reference

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

Part No: 819–2140–12
April 2006

060424@14774

Contents

Preface 23

1 About the Access Manager C APIs 29
Summary of C Header Files 29
Summary of C Code Samples 30
Required C Libraries 31
Solaris Platform 31
Linux Platform 31
Windows Platform 31

2 Type and Structure Reference 33
am_auth_callback 33
Syntax 34
Members 34
am_auth_choice_callback 35
Syntax 35
Members 35
Details 36
am_auth_confirmation_callback_info 36
Syntax 36
Members 36
Details 37
am_auth_language_callback_info 37
Syntax 37
Members 38
am_auth_locale 38
Syntax 38
Members 38

Details 39
am_auth_name_callback_info 39
Syntax 39
Members 39
Details 40
am_auth_password_callback_info 40
Syntax 40
Members 40
Details 40
am_auth_text_input_callback_info 41
Syntax 41
Members 41
Details 41
am_auth_text_output_callback_info 42
Syntax 42
Members 42
Details 42
am_log_record 42
Syntax 42
Members 43
Details 43
am_map_t 43
Syntax 43
Members 43
Details 43
am_map_entry_iter 44
Syntax 44
Members 44
Details 44
am_map_value_iter 44
Syntax 44
Members 44
Details 44
am_policy_result 45
Syntax 45
Members 45
Details 45

am_resource_traits 45
Syntax 46
Members 46
Details 48
am_string_set_t 48
Syntax 48
Members 48
Details 48

3 Authentication Functions 49
am_auth_abort() 49
Syntax 49
Parameters 50
Returns 50
am_auth_create_auth_context() 50
Syntax 50
Parameters 50
Returns 51
am_auth_destroy_auth_context() 51
Syntax 51
Parameters 51
Returns 51
am_auth_get_module_instance_names() 51
Syntax 52
Parameters 52
Returns 52
Details 52
am_auth_get_organization_name() 52
Syntax 53
Parameters 53
Returns 53
am_auth_get_sso_token_id() 53
Syntax 53
Parameters 53
Returns 53
am_auth_get_sso_token_id() 54

Syntax 54
Parameters 54
Returns 54
am_auth_has_more_requirements() 54
Syntax 54
Parameters 55
Returns 55
Details 55
am_auth_init() 55
Syntax 55
Parameters 55
Returns 55
am_auth_login() 56
Syntax 56
Parameters 56
Returns 56
am_auth_logout() 57
Syntax 57
Parameters 57
Returns 57
am_auth_num_callbacks() 57
Syntax 57
Parameters 57
Returns 58
am_auth_submit_requirements() 58
Syntax 58
Parameters 58
Returns 58

4 Logging Functions 59
am_log_add_module() 60
Syntax 60
Parameters 60
Returns 60
Details 60
am_log_flush_remote_log() 60

Syntax 61
Parameters 61
Returns 61
am_log_init() 61
Syntax 61
Parameters 61
Returns 62
am_log_is_level_enabled() 62
Syntax 62
Parameters 62
Returns 62
am_log_log() 63
Syntax 63
Parameters 63
Returns 63
Details 63
am_log_log_record() 63
Syntax 64
Parameters 64
Returns 64
am_log_record_add_loginfo() 64
Syntax 64
Parameters 64
Returns 65
am_log_record_create() 65
Syntax 65
Parameters 65
Returns 65
am_log_record_destroy() 66
Syntax 66
Parameters 66
Returns 66
am_log_record_populate() 66
Syntax 66
Parameters 67
Returns 67
am_log_record_set_log_level() 67

Syntax 67
Parameters 67
Returns 67
am_log_record_set_log_message() 68
Syntax 68
Parameters 68
Returns 68
am_log_record_set_loginfo_props() 68
Syntax 68
Parameters 69
Returns 69
Details 69
am_log_set_levels_from_string() 69
Syntax 69
Parameters 69
Returns 70
Details 70
am_log_set_log_file() 70
Syntax 70
Parameters 70
Returns 70
Details 71
am_log_set_module_level() 71
Syntax 71
Parameters 71
Returns 71
am_log_set_remote_info() 72
Syntax 72
Parameters 72
Returns 72
am_log_vlog() 73
Syntax 73
Parameters 73
Returns 73
Details 73

5 Map Functions 75
am_map_clear() 76
Syntax 76
Parameters 76
Returns 76
am_map_copy() 76
Syntax 76
Parameters 76
Returns 77
am_map_create() 77
Syntax 77
Parameters 77
Returns 77
am_map_destroy() 78
Syntax 78
Parameters 78
Returns 78
Details 78
am_map_entry_iter_destroy() 78
Syntax 79
Parameters 79
am_map_entry_iter_get_first_value() 79
Syntax 79
Parameters 79
Returns 79
Details 80
am_map_entry_iter_get_key() 80
Syntax 80
Parameters 80
Returns 80
Details 80
am_map_entry_iter_get_values() 81
Syntax 81
Parameters 81
Returns 81
Details 81
am_map_entry_iter_is_entry_valid() 82

Syntax 82
Parameters 82
Returns 82
am_map_entry_iter_next() 82
Syntax 82
Parameters 82
Returns 83
am_map_erase() 83
Syntax 83
Parameters 83
Returns 83
am_map_find() 83
Syntax 84
Parameters 84
Returns 84
Details 84
am_map_find_first_value() 85
Syntax 85
Parameters 85
Returns 85
Details 85
am_map_get_entries() 85
Syntax 86
Parameters 86
Returns 86
Details 86
am_map_insert() 86
Syntax 87
Parameters 87
Returns 87
Details 87
am_map_size() 87
Syntax 88
Parameters 88
Returns 88
am_map_entry_iter_destroy() 88
Syntax 88

Parameters 88
Details 88
am_map_value_iter_get() 89
Syntax 89
Parameters 89
Returns 89
am_map_value_iter_is_value_valid() 89
Syntax 89
Parameters 89
Returns 90

6 Policy Functions 91
am_policy_compare_urls() 91
Syntax 91
Parameters 92
Returns 92
Details 92
am_policy_destroy() 92
Syntax 92
Parameters 92
Returns 93
Details 93
am_policy_evaluate() 93
Syntax 93
Parameters 93
Returns 94
Details 94
am_policy_get_url_resource_root() 94
Syntax 94
Parameters 94
Returns 95
Details 95
am_policy_init() 95
Syntax 95
Parameters 95
Returns 95

Details 96
am_policy_is_notification_enabled() 96
Syntax 96
Parameters 96
Returns 96
am_policy_notify() 96
Syntax 96
Parameters 97
Returns 97
am_policy_resource_canonicalize() 97
Syntax 97
Parameters 97
am_policy_resource_has_patterns() 98
Syntax 98
Parameters 98
Returns 98
am_policy_result_destroy() 98
Syntax 98
Parameters 98
Returns 99
am_policy_service_init() 99
Syntax 99
Parameters 99
Returns 99

7 Properties Functions 101
am_properties_copy() 102
Syntax 102
Parameters 102
Returns 102
Details 102
am_properties_create() 103
Syntax 103
Parameters 103
Returns 103
Details 103

am_properties_destroy() 103
Syntax 104
Parameters 104
Returns 104
Details 104
am_properties_get() 104
Syntax 104
Parameters 105
Returns 105
Details 105
am_properties_get_boolean() 105
Syntax 105
Parameters 106
Returns 106
Details 106
am_properties_get_boolean_with_default() 106
Syntax 106
Parameters 106
Returns 107
Details 107
am_properties_get_entries() 107
Syntax 107
Parameters 107
Returns 107
Details 108
am_properties_get_signed() 108
Syntax 108
Parameters 108
Returns 108
Details 108
am_properties_get_signed_with_default() 109
Syntax 109
Parameters 109
Returns 109
Details 109
am_properties_get_unsigned() 109
Syntax 110

Parameters 110
Returns 110
Details 110
am_properties_get_unsigned_with_default() 110
Syntax 110
Parameters 110
Returns 111
Details 111
am_properties_get_with_default() 111
Syntax 111
Parameters 111
Details 111
am_properties_is_set() 112
Syntax 112
Parameters 112
Returns 112
am_properties_iter_destroy() 112
Syntax 112
Parameters 113
Returns 113
am_properties_iter_get_key() 113
Syntax 113
Parameters 113
Returns 113
am_properties_iter_get_value() 114
Syntax 114
Parameters 114
Returns 114
am_properties_load() 114
Syntax 114
Parameters 114
Returns 115
Details 115
am_properties_set() 115
Syntax 115
Parameters 115
Returns 116

Details 116
am_properties_store() 116
Syntax 116
Parameters 116
Returns 116

8 Single Sign-On Functions 117
am_sso_add_listener() 118
Syntax 118
Parameters 118
Returns 118
Details 118
am_sso_add_sso_token_listener() 119
Syntax 119
Parameters 119
Returns 120
Details 120
am_sso_create_sso_token_handle() 120
Syntax 120
Parameters 121
Returns 121
am_sso_destroy_sso_token_handle() 121
Syntax 121
Parameters 121
Returns 122
Details 122
am_sso_get_auth_level() 122
Syntax 122
Parameters 122
Returns 122
am_sso_get_auth_type() 123
Syntax 123
Parameters 123
Returns 123
am_sso_get_host() 123
Syntax 123

Parameters 123
Returns 124
am_sso_get_idle_time 124
Syntax 124
Parameters 124
Returns 124
am_sso_get_max_idle_time() 124
Syntax 124
Parameters 125
Returns 125
am_sso_get_max_session_time() 125
Syntax 125
Parameters 125
Returns 125
am_sso_get_principal() 125
Syntax 126
Returns 126
am_sso_get_principal_set() 126
Syntax 126
Parameters 126
Returns 126
am_sso_get_property() 127
Syntax 127
Parameters 127
Returns 127
am_sso_get_sso_token_id() 127
Syntax 127
Parameters 128
Returns 128
am_sso_get_time_left() 128
Syntax 128
Parameters 128
Returns 128
Details 128
am_sso_init() 129
Syntax 129
Parameters 129

Returns 129
Details 129
am_sso_invalidate_token() 129
Syntax 129
Parameters 130
Returns 130
Details 130
am_sso_is_valid_token() 130
Syntax 130
Parameters 131
Returns 131
Details 131
am_sso_refresh_token() 131
Syntax 131
Parameters 131
Returns 131
Details 132
am_sso_remove_listener() 132
Syntax 132
Parameters 132
Returns 132
Details 133
am_sso_remove_sso_token_listener() 133
Syntax 133
Parameters 133
Returns 133
Details 134
am_sso_set_property() 134
Syntax 134
Parameters 134
Returns 134
Details 135
am_sso_validate_token() 135
Syntax 135
Parameters 135
Returns 135
Details 136

9 Web Functions 137
am_web_clean_post_urls() 138
Syntax 138
Parameters 138
Returns 138
am_web_cleanup() 138
Syntax 138
Parameters 139
Returns 139
am_web_create_post_page() 139
Syntax 139
Parameters 139
Returns 139
am_web_create_post_preserve_urls() 140
Syntax 140
Parameters 140
Returns 140
Details 140
am_web_free_memory() 140
Syntax 140
Parameters 141
Returns 141
am_web_get_agent_server_host() 141
Syntax 141
Parameters 141
Returns 141
am_web_get_agent_server_port() 141
Syntax 142
Parameters 142
Returns 142
am_web_get_cookie_name() 142
Syntax 142
Parameters 142
Returns 142
am_web_get_notification_url() 143
Syntax 143
Parameters 143

Returns 143
am_web_get_parameter_value() 143
Syntax 143
Parameters 143
Returns 144
am_web_get_redirect_url() 144
Syntax 144
Parameters 144
Returns 145
Details 145
am_web_get_token_from_assertion() 145
Syntax 145
Parameters 145
Returns 146
am_web_handle_notification() 146
Syntax 146
Parameters 146
Returns 146
Details 146
am_web_http_decode() 147
Syntax 147
Parameters 147
Returns 147
am_web_init() 147
Syntax 147
Parameters 147
Returns 148
am_web_is_access_allowed() 148
Syntax 148
Parameters 148
Returns 149
am_web_is_cdsso_enabled() 149
Syntax 149
Parameters 149
Returns 149
am_web_is_debug_on() 150
Syntax 150

Parameters 150
Returns 150
am_web_is_in_not_enforced_ip_list() 150
Syntax 150
Parameters 150
Returns 151
am_web_is_in_not_enforced_list() 151
Syntax 151
Parameters 151
Returns 151
am_web_is_max_debug_on() 151
Syntax 152
Parameters 152
Returns 152
am_web_is_notification() 152
Syntax 152
Parameters 152
Returns 152
am_web_is_postpreserve_enabled() 153
Syntax 153
Parameters 153
Returns 153
am_web_is_valid_fqdn_url() 153
Syntax 153
Parameters 153
Returns 154
am_web_log_always() 154
Syntax 154
Parameters 154
Returns 154
am_web_log_auth() 154
Syntax 154
Parameters 155
Returns 155
am_web_log_debug() 155
Syntax 155
Parameters 155

Returns 155
am_web_log_error() 156
Syntax 156
Parameters 156
Returns 156
am_web_log_info() 156
Syntax 156
Parameters 156
Returns 157
am_web_log_max_debug() 157
Syntax 157
Parameters 157
Returns 157
am_web_log_warning() 157
Syntax 157
Parameters 157
Returns 158
am_web_postcache_data_cleanup() 158
Syntax 158
Parameters 158
Returns 158
am_web_postcache_insert() 158
Syntax 158
Parameters 159
Returns 159
am_web_postcache_lookup() 159
Syntax 159
Parameters 159
Returns 159
am_web_postcache_remove() 160
Syntax 160
Parameters 160
Returns 160
am_web_remove_parameter_from_query() 160
Syntax 160
Parameters 160
Returns 161

10 Miscellaneous Functions 163
am_cleanup() 163
Syntax 163
Parameters 163
Returns 164
Details 164
am_notify() 164
Syntax 164
Parameters 164
Returns 165
Details 165
am_string_set_allocate() 165
Syntax 165
Parameters 165
Returns 165
am_string_set_destroy() 166
Syntax 166
Parameters 166
Returns 166
am_status_to_name() 166
Syntax 166
Parameters 166
Returns 167
am_status_to_string() 167
Syntax 167
Parameters 167
Returns 167
Details 167
am_http_cookie_encode() 168
Syntax 168
Parameters 168
Returns 168
am_http_cookie_decode() 168
Syntax 168
Parameters 169
Returns 169

Preface

Sun Java™ System Access Manager is a component of the Sun Java Enterprise System (Java ES), a set of software components that provide services needed to support enterprise applications distributed across a network or Internet environment. The *Sun Java System Access Manager 7 2005Q4 C API Reference* provides a listing of APIs you can use to enable C applications to access the Access Manager services. The reference includes function descriptions and syntax.

Revision History

The following table shows the Access Manager 7 2005Q4 Release Notes revision history.

TABLE P–1 Revision History

Date	Part Number	Description of Changes
January 2006	819-2140-11	Added note in section "Summary of C Code Samples" on page 30.
April 2006	819-2140-12	Added Chapter 10.

Before You Read This Book

This book is intended for use by IT administrators and software developers who implement a web access platform using Sun Java System servers and software. Readers of this guide should be familiar with the following concepts and technologies:

- Access Manager technical concepts as described in the *Sun Java System Access Manager 7 2005Q4 Technical Overview*
- Deployment platform: Solaris or Linux operating system
- Web container that will run Access Manager: Sun Java System Application Server, Sun Java System Web Server, BEA WebLogic, or IBM WebSphere Application Server
- Technical concepts: Lightweight Directory Access Protocol (LDAP), Java technology, JavaServer Pages (JSP) technology, HyperText Transfer Protocol (HTTP), HyperText Markup Language (HTML), and eXtensible Markup Language (XML)

Related Books

Related documentation is available as follows:

- "Access Manager Core Documentation" on page 24
- "Sun Java Enterprise System Product Documentation" on page 25

Access Manager Core Documentation

The Access Manager core documentation set contains the following titles:

- The *Sun Java System Access Manager 7 2005Q4 Release Notes* will be available online after the product is released. It gathers an assortment of last-minute information, including a description of what is new in this current release, known problems and limitations, installation notes, and how to report issues with the software or the documentation.
- The *Sun Java System Access Manager 7 2005Q4 Technical Overview* provides an overview of how Access Manager components work together to consolidate access control functions, and to protect enterprise assets and web-based applications. It also explains basic Access Manager concepts and terminology.
- The *Sun Java System Access Manager 7 2005Q4 Deployment Planning Guide* provides planning and deployment solutions for Sun Java System Access Manager based on the solution life cycle.
- The *Sun Java System Access Manager 7 2005Q4 Performance Tuning Guide* provides information on how to tune Access Manager and its related components for optimal performance.
- The *Sun Java System Access Manager 7 2005Q4 Administration Guide* describes how to use the Access Manager console as well as manage user and service data via the command line interface.
- The *Sun Java System Access Manager 7 2005Q4 Federation and SAML Administration Guide* provides information about the Federation module based on the Liberty Alliance Project specifications. It includes information on the integrated services based on these specifications, instructions for enabling a Liberty-based environment, and summaries of the application programming interface (API) for extending the framework.
- The *Sun Java System Access Manager 7 2005Q4 Developer's Guide* offers information on how to customize Access Manager and integrate its functionality into an organization's current technical infrastructure. It also contains details about the programmatic aspects of the product and its API.
- The *Sun Java System Access Manager 7 2005Q4 Java API Reference* (this guide) provides information about the implementation of Java packages in Access Manager.
- The *Sun Java System Access Manager Policy Agent 2.2 User's Guide* provides an overview of the policy functionality and the policy agents available for Access Manager.

Updates to the *Sun Java System Access Manager 7 2005Q4 Release Notes* and links to modifications of the core documentation can be found on the Access Manager page at the Sun Java Enterprise System documentation web site. Updated documents will be marked with a revision date.

Sun Java Enterprise System Product Documentation

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

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

Related Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – 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.

Documentation, Support, and Training

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

Typographic Conventions

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

TABLE P–2 Typographic Conventions

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

Shell Prompts in Command Examples

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

TABLE P–3 Shell Prompts

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

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 book or at the top of the document.

For example, the title of this book is *Sun Java System Access Manager 7 2005Q4 C API Reference*, and the part number is 819-2140–10.

◆ ◆ ◆ CHAPTER 1

About the Access Manager C APIs

This chapter provides information for locating the files you need to use the C APIs. Topics included in this chapter are:

- "Summary of C Header Files" on page 29
- "Summary of C Code Samples" on page 30
- "Required C Libraries" on page 31

Summary of C Header Files

By default, C header files are installed in the following directory:
/AccessManager-base-dir`/SUNWam/include`

The `/include` directory contains the following C header files:

`am.h`	General utility routines provided by the Access Manager library.
`am_auth.h`	Public functions for developing custom authentication modules.
`am_log.h`	Public functions for logging on the local system or on the Access Manager server.
`am_map.h`	Public functions for creating, destroying, and manipulating the map objects used by Access Manager.
`am_notify.h`	Public function for implementing notifications.
`am_policy.h`	Public functions for using Access Manager policy objects.
`am_properties.h`	Properties map used by clients of the Access Manager Client APIs.
`am_sso.h`	Public functions for implementing Single Sign-on (SSO) in Access Manager.
`am_string_set.h`	Common types and macros provided by the Access Manager.
`am_types.h`	Common types and macros provided by Access Manager.

`am_utils.h`	This is an unsupported, Early Access version of utility functions. Functions and data structures may change without backward compatibility.
`am_web.h`	Public functions intended for use by only Access Manager web agents.

Summary of C Code Samples

Access Manager provides code samples that demonstrate how you can use the C APIs to connect your C applications to the Access Manager framework. By default, the code samples are installed in the following directory:

/AccessManager-base-dir`/SUNWam/samples/csdk`

The `csdk` directory contains the following files:

`am_auth_test.c`	Demonstrates the basic usage of Authentication SDK APIs you can use to login to an Identity Server.
`am_log_test.c`	Demonstrates the basic usage of Logging SDK APIs you can use to log a message to the Identity Server logs.
`am_policy_test.c`	Demonstrates the basic usage of Policy SDK APIs you can use to evaluate policy for specified resources.

Note – Before running the test, be sure the password for the property `com.sun.am.poicy.am.password` is in clear text. The sample `am_policy_test.c` does not decrypt the password before authenticating with Access Manager. Since it is only a sample, when running `am_policy_test.c` for testing purposes, having the password in clear text poses no security risk. Example:

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

`am_sso_test.c`	Demonstrates the basic usage of SSO SDK APIs you can use to perform session operations.
`apache_agent.c`	Demonstrates how the you can use Policy APIs to build a Web Agent for the Apache Web Server. This is a sample Web Agent and is not intended to serve as a real Web Agent.
`Makefile`	Makefile for building a sample agent.
`README.TXT`	Provides detailed instructions for building and executing sample programs.

Required C Libraries

You can run the sample programs by launching the generated executables on the command line. Be sure to set the library path appropriately for the platform you are using.

Solaris Platform

Set the `LD_LIBRARY_PATH` environment variable to include the following library directories:`/usr/lib/mps:/opt/SUNWam/lib:/usr/lib:/usr/ucblib`
which contain:
`libamsdk.so`, `libxml2.so`, `libssl3.so`, `libnss3.so`, `libplc4.so`, `libplds4.so`, `libnspr4.so`, and `libucb.so`.

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

Linux Platform

Set the `LD_LIBRARY_PATH` environment variable to include the directory:
AccessManager-base-dir`/agent/lib`
which contains:
`libamsdk.so`, `libxml2.so`,`libssl3.so`, `libnss3.so`, `libplc4.so`, `libplds4.so` and `libnspr4.so`.

Windows Platform

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

◆ ◆ ◆ CHAPTER 2

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. 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" on page 33
- "am_auth_choice_callback" on page 35
- "am_auth_confirmation_callback_info" on page 36
- "am_auth_language_callback_info" on page 37
- "am_auth_locale" on page 38
- "am_auth_name_callback_info" on page 39
- "am_auth_password_callback_info" on page 40
- "am_auth_text_input_callback_info" on page 41
- "am_auth_text_input_callback_info" on page 41
- "am_log_record" on page 42
- "am_map_entry_iter" on page 44
- "am_map_value_iter" on page 44
- "am_policy_result" on page 45
- "am_resource_traits" on page 45
- "am_string_set_t" on page 48

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_t    confirmation_callback;
        am_auth_language_callback_t language_callback;
        am_auth_name_callback_t name_callback;
        am_auth_password_callback_t password_callback;
        am_auth_text_input_callback_t text_input_callback;
        am_auth_text_output_callback_t text_output_callback;
    } callback_info;
} am_auth_callback_t;
```

Members

This structure has the following members:

`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 members 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;
```

Members

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 members:

`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;
```

Members

This structure has the following members:

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;
```

Members

This structure has the following members:

`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;
```

Members

This structure has the following members:

`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;
```

Members

This structure has the following members:

`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;
```

Members

This structure has the following members:

`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;
```

Members

This structure has the following members:

`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;
```

Members

This structure has the following members:

`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;
```

Members

This is an opaque structure and therefore has no members 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;
```

Members

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

Details

This function creates an instance of `am_map_t` structure and returns the pointer to the structure to the caller.

Memory Concerns: You should free the allocated structure by calling `am_map_destroy`.

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;
```

Members

This is an opaque structure and therefore has no members 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;
```

Members

This is an opaque structure and therefore has no members 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 evaluation 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;
```

Members

This structure has the following members:

`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_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;
```

Members

This structure has the following members:

`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.

Inputs:

`rsrc_traits` - the resource traits structure to use.

`policy_res_name` - name of a resource in the policy tree.

`resource_name` - name of the resource in policy evaluation.

`use_patterns` - whether to use or recognize patterns when comparing resources.

Returns:

Return one of `AM_SUB_RESOURCE_MATCH`, `AM_EXACT_MATCH` , `AM_SUPER_RESOURCE_MATCH`, `AM_NO_MATCH`, or `AM_EXACT_PATTERN_MATCH`.

Example:

`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.

Inputs:

`resource_name` - name of the resource.

Returns:

True if `resource_name` has patterns and false otherwise.

Example:

`am_policy_resource_has_patterns` can be used for URL resources.

`boolean_t (*get_resource_root) ( const char *resource_nam, char *root_resource_name, size_t buflength);`
A function to get the root of a resource.

Inputs:

`Resource_name` - name of the resource.

`Root_resource_name` - a buffer to contain the name of the resource root.

`Buflength` - length of the `root_resource_name` buffer passed to this function.

Returns:

True if the name of the resource root was successfully inserted into the given `root_resource_name` buffer, false otherwise.

Examples:

`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 separator. For URLs '/' should be used as the separator.

`void (*canonicalize) (const char *resource, char **c_resource);`
A function to canonicalize a resource name.

Inputs:

`resource` - the resource name.

Outputs:

`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) (void *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()`.

This field cannot be set to null.

Inputs:

The `resource_str - the` string to be freed.

Examples:

Method `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;
```

Members

This structure has the following members:

`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.

◆ ◆ ◆ CHAPTER 3

Authentication Functions

This chapter provides a reference to the public functions you can use in developing custom authentication modules for Sun Java™ System Access Manager. 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 49
- "am_auth_create_auth_context()" on page 50
- "am_auth_destroy_auth_context()" on page 51
- "am_auth_get_module_instance_names()" on page 51
- "am_auth_get_organization_name()" on page 52
- "am_auth_get_sso_token_id()" on page 53
- "am_auth_has_more_requirements()" on page 54
- "am_auth_init()" on page 55
- "am_auth_login()" on page 56
- "am_auth_logout()" on page 57
- "am_auth_num_callbacks()" on page 57
- "am_auth_submit_requirements()" on page 58

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:

auth_ctx	Handle of the auth context.

Returns

This function returns am_status_t with one of the following values:

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:

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: http://pride.red.iplanet.com:58080/amserver 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:

`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:

`auth_ctx`	Handle of the `auth context` to be destroyed.

Returns

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

`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:

`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:

`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 Authentication 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:

`auth_ctx` Handle of the `auth context`.

Returns

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

Zero-terminated string representing the organization
: When user successfully 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:

`uth_ctx` Handle of the `auth context`.

Returns

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

Zero-terminated string representing the organization
When user successfully 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:

`auth_ctx` Handle of the `auth context`.

Returns

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

Zero-terminated string representing the organization.
When user successfully 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:

`auth_ctx` Handle of the `auth context`.

Returns

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

`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:

`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:

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:

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:

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

am_auth_logout()

Logs out the user.

Syntax

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

Parameters

This function takes the following parameter:

`auth_ctx` Handle of the `auth context`.

Returns

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

`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:

`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:

`auth_ctx` Handle of the `auth context`.

Returns

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

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

◆ ◆ ◆ CHAPTER 4

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. 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()" on page 60
- "am_log_flush_remote_log()" on page 60
- "am_log_init()" on page 61
- "am_log_is_level_enabled()" on page 62
- "am_log_log()" on page 63
- "am_log_log_record()" on page 63
- "am_log_record_add_loginfo()" on page 64
- "am_log_record_create()" on page 65
- "am_log_record_destroy()" on page 66
- "am_log_record_populate()" on page 66
- "am_log_record_set_log_level()" on page 67
- "am_log_record_set_log_message()" on page 68
- "am_log_record_set_loginfo_props()" on page 68
- "am_log_set_levels_from_string()" on page 69
- "am_log_set_log_file()" on page 70
- "am_log_set_module_level()" on page 71
- "am_log_set_remote_info()" on page 72
- "am_log_vlog()" on page 73

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:

`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:

`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:

`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, authentication, 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, authentication, 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:

`log_init_params` Properties to initialize the log module with.

Returns

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

`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:

`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:

`!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:

`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:

`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:

`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:

`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:

`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:

`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:

`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 parameter:

record	Opaque handle to the log record to destroy.

Returns

This function returns am_status_t with one of the following values:

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:

`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:

`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:

`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:

`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:

`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:

`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:

`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:

`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 members 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:

`module_level_string`	List of modules to set.

Returns

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

`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:

`name`	Name of the file in which to record logging messages.

Returns

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

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:

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:

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:

`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:

`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:

`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.

◆ ◆ ◆ CHAPTER 5

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 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_clear()" on page 76
- "am_map_copy()" on page 76
- "am_map_create()" on page 77
- "am_map_destroy()" on page 78
- "am_map_entry_iter_destroy()" on page 78
- "am_map_entry_iter_get_first_value()" on page 79
- "am_map_entry_iter_get_key()" on page 80
- "am_map_entry_iter_get_values()" on page 81
- "am_map_entry_iter_is_entry_valid()" on page 82
- "am_map_entry_iter_next()" on page 82
- "am_map_erase()" on page 83
- "am_map_find_first_value()" on page 85
- "am_map_get_entries()" on page 85
- "am_map_insert()" on page 86
- "am_map_size()" on page 87
- "am_map_entry_iter_destroy()" on page 88
- "am_map_value_iter_get()" on page 89
- "am_map_value_iter_is_value_valid()" on page 89

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 parameter:

`map` The handle for the map object to be modified.

Returns

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

`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:

`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:

`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.

Details

This function creates an instance of `am_map_t` structure, copies all the elements in `source_map` into the newly created structure and assigns it to `map_ptr`. It does not alter the contents of `source_map`.

Memory Concerns: The caller must make sure not to pass a `map_ptr` which as a valid `am_map_t` structure, otherwise the reference will be lost. The caller must destroy `map_ptr` after usage by calling `am_map_destroy`.

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:

`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:

`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:

`map` The handle for the map object to be destroyed. The handle may be NULL.

Returns

This function returns one of the following values:

`AM_SUCCESS`	If the destroy operation was successfully performed.
`AM_NO_MEMORY`	If there was an internal memory operation error.
`AM_INVALID_ARGUMENT`	If the address of `map_ptr` or `source_map` is invalid.

Details

This function destroys an instance of `am_map_t` structure which is pointed by `map_ptr`.

Memory Concerns: Care must be taken that `map_ptr` was not freed before by calling `am_map_destroy` or by erroneously calling the system void free `(void *)` function.

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 parameter:

`entry_iter` The handle for the key iterator object to be destroyed. The handle may be NULL.

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 parameter:

`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 If the operation is successful, returns the first associated value of this iterator. The order of insertion into the map does not guarantee the value returned.

`NULL` If the specified iterator is NULL or does not reference a valid entry or the entry does not have any associated values.

Details

This function destroys the `am_map_entry_iterator_t` passed to it.

Memory Concerns: Caller must be sure that this function is not called multiple times on the same `am_map_entry_iter_t`.

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:

`entry_iter` The handle for the entry iterator object to be examined.

Returns

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

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

`key` Otherwise

Details

This function returns the key of this key-value pair entry iterator.

Memory Concerns: Caller must not modify or free the return value.

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:

`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:

`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.

Details

This function returns an `am_map_value_iter_t` that enumerates over the values associated with `am_map_entry_iter_t`.

Memory Concerns After the use of `value_iter_t` the caller must call `am_map_value_iter_destroy`.

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 parameter:

`entry_iter` The handle for the entry iterator object to be examined.

Returns

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

`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:

`entry_iter` The handle for the entry iterator object to be modified.

Returns

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

`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:

`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:

`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:

`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:

`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

This function takes a key and returns an iterator that iterates over the values associated with the key.

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.

Memory Concerns: At the end of usage of `value_iter_ptr`, the caller must call `am_map_value_iter_destroy` with the iterators pointer.

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:

`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:

`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.

Details

This function takes a key and returns the first value associated with the key.

Memory Concerns: Caller must not modify or free the return value.

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:

`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:

`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

This function extracts an iterator pointer that can be used to iterate over the key value pairs stored in this table.

Memory Concerns: The iterator pointer passed in must not have non-destroyed iterators assigned to them. The caller, in future must call `am_map_entry_iter_destroy` to destroy the iterator instance.

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:

`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:

`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:

`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 parameter:

`entry_iter` The handle for the key iterator object to be destroyed. The handle may be NULL.

Details

This function destroys the `am_map_entry_iterator_t` passed to it.

Memory Concerns: Caller must be sure that this function is not called multiple times on the same `am_map_entry_iter_t`.

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:

`value_iter` The handle for the value iterator object to be examined.

Returns

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

`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:

`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:

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

`!0` Otherwise

◆ ◆ ◆ CHAPTER 6

Policy Functions

This chapter provides a reference to the public functions for using Sun Java™ System Access Manager 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 91
- "am_policy_destroy()" on page 92
- "am_policy_evaluate()" on page 93
- "am_policy_get_url_resource_root()" on page 94
- "am_policy_init()" on page 95
- "am_policy_is_notification_enabled()" on page 96
- "am_policy_notify()" on page 96
- "am_policy_resource_canonicalize()" on page 97
- "am_policy_resource_has_patterns()" on page 98
- "am_policy_result_destroy()" on page 98
- "am_policy_service_init()" on page 99

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:

`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:

`policy` Opaque handle to the policy service to destroy.

Returns

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

`AM_SUCCESS` If the call was successful.

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

Details

This function destroys a policy service instance. Memory Concerns: Caller must call make sure the same service instance not be destroyed more than once.

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:

`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:

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

Details

This function destroys a policy service instance. Memory Concerns: After using the results the caller must call am_policy_result_destroy on the policy_result to cleanup the memory allocated by the evaluation operation. am_map_destroy must also be called on response and env_parameter_map after their respective usage scope.

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:

`AM_TRUE` Successful root extraction.

`AM_FALSE` Otherwise

Details

This function is takes a URL and extracts a root of the URL. For example, `http://www.sun.com/index.html` will return `http://www.sun.com/` and `http://www.sun.com:8080/index.html` will return `http://www.sun.com:8080/`. Memory Concerns: In an implementation for a different resource other than URLs, the service writer implementing this function must make accurate judgement about the minimum size of `resourceRoot`.

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:

`properties` The properties to initialize the policy service with.

Returns

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

`AM_SUCCESS` If the call was successful.

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

Details

This function initializes a policy service instance. Memory Concerns: Caller must call `am_policy_destroy` structure or free the memory.

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:

`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:

`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:

`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:

`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:

`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()`.

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 parameter:

`resource_name` Name of the resource.

Returns

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

`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:

`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:

`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:

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

◆ ◆ ◆ CHAPTER 7

Properties Functions

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

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

- "am_properties_copy()" on page 102
- "am_properties_create()" on page 103
- "am_properties_destroy()" on page 103
- "am_properties_get()" on page 104
- "am_properties_get_boolean()" on page 105
- "am_properties_get_boolean_with_default()" on page 106
- "am_properties_get_entries()" on page 107
- "am_properties_get_signed()" on page 108
- "am_properties_get_signed_with_default()" on page 109
- "am_properties_get_unsigned()" on page 109
- "am_properties_get_unsigned_with_default()" on page 110
- "am_properties_get_with_default()" on page 111
- "am_properties_is_set()" on page 112
- "am_properties_iter_destroy()" on page 112
- "am_properties_iter_get_key()" on page 113
- "am_properties_iter_get_value()" on page 114
- "am_properties_load()" on page 114
- "am_properties_set()" on page 115
- "am_properties_store()" on page 116

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:

`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:

`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.

Details

Creates an instance of `am_properties_t` and assigns it to `properties_ptr`. The function copies all the elements in the `source_ptr` to `properties_ptr`. The `source_ptr` is not affected during this operation.

Memory Concerns: After the usage of the instance `properties_ptr` the caller must call `am_properties_destroy` to clean up the allocated memory. The removal of any item in either structures do not affect the other.

am_properties_create()

Creates an empty properties object.

Syntax

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

Parameters

This function takes the following parameters:

`properties_ptr` 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:

`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.

Details

Creates an instance of `am_properties_t` and assigns it to `properties_ptr`.

Memory Concerns: After the usage of the instance the caller must call `am_properties_destroy` to clean up the allocated memory.

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:

`properties` The handle for the properties object to be destroyed.

Returns

This function returns one of the following values:

`AM_SUCCESS`	If the operation was successful.
`AM_INVALID_ARGUMENT`	If properties argument is NULL.

Details

Destroys an instance of `am_properties_t`.

Memory Concerns: Caller must make sure not to pass the same instance of `am_properties_t` to be destroyed more than once. After calling this function it is advised that the caller initializes properties to NULL.

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:

`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:

`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.

Details

This function checks if the key is present is the properties instance and returns its value.

Memory Concerns: Caller must not modify the `value_ptr` structure or free the memory.

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()" on page 104.

Returns

- The unparsed string form of the value associated with the specified key. See "am_properties_get()" on page 104.
- A value stored in `value_ptr` with one of the following values.

`!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()" on page 104.

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()" on page 104.

Returns

- The unparsed string form of the value associated with the specified key. See "am_properties_get()" on page 104.
- A value stored in `value_ptr` with one of the following values.

`!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()" on page 104.

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()" on page 104 .

Syntax

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

Parameters

This function takes the following parameters:

`properties`	The 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:

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()" on page 104.

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()" on page 104.

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()" on page 104.

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()" on page 104.

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()" on page 104.

am_properties_get_unsigned()

See "am_properties_get()" on page 104.

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()” on page 104.

Returns

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

Details

See “am_properties_get()” on page 104.

am_properties_get_unsigned_with_default()

See “am_properties_get()” on page 104.

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()” on page 104.

Returns

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

Details

See "am_properties_get()" on page 104.

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

This function takes the following parameters:

`properties` The `am_properties_t` instance from which the keys value needs to be extracted. key The key whose value will be returned.

`default_value` The value to be returned in case of any error condition. `value_ptr` The value pointer to which the value will be assigned to. This is an output parameter. Returns Return values may be ignored.

Details

This function checks if the key is present is the properties instance. If the key is not present, the function returns the default value passed in. Otherwise it returns the value of the key.

Memory Concerns: Caller must not modify the `value_ptr` `structure` or free the memory.

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:

`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:

`!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:

`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:

`properties_iter` The handle for the properties iterator object to be examined.

Returns

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

`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:

`properties_iter` The handle for the properties iterator object to be examined.

Returns

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

`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:

`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:

`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:

`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:

`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:

`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:

`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.

◆ ◆ ◆ CHAPTER 8

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. 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()" on page 118
- "am_sso_add_sso_token_listener()" on page 119
- "am_sso_create_sso_token_handle()" on page 120
- "am_sso_destroy_sso_token_handle()" on page 121
- "am_sso_get_auth_level()" on page 122
- "am_sso_get_auth_level()" on page 122
- "am_sso_get_auth_type()" on page 123
- "am_sso_get_host()" on page 123
- "am_sso_get_idle_time" on page 124
- "am_sso_get_max_idle_time()" on page 124
- "am_sso_get_max_session_time()" on page 125
- "am_sso_get_principal()" on page 125
- "am_sso_get_principal_set()" on page 126
- "am_sso_get_property()" on page 127
- "am_sso_get_sso_token_id()" on page 127
- "am_sso_get_time_left()" on page 128
- "am_sso_init()" on page 129
- "am_sso_invalidate_token()" on page 129
- "am_sso_is_valid_token()" on page 130
- "am_sso_refresh_token()" on page 131
- "am_sso_remove_listener()" on page 132
- "am_sso_remove_sso_token_listener()" on page 133
- "am_sso_set_property()" on page 134
- "am_sso_validate_token()" on page 135

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:

`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:

`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 UR 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:

`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:

`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:

`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:

`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 parameter:

`sso_token_handle`	SSO token handle to be de-allocated.

Returns

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

`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 authentication 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:

`sso_token_handle`	The SSO token handle.

Returns

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

am_sso_get_auth_type()

Gets the authentication 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 parameter:

`sso_token_handle` The SSO token handle.

Returns

This function returns the authentication 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 parameter:

`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 parameter:

`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 maximum 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:

`sso_token_handle` The SSO token handle.

Returns

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

am_sso_get_max_session_time()

Gets the maximum session time for this session.

Syntax

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

Parameters

This function takes the following parameters:

`sso_token_handle` The SSO token handle.

Returns

This function returns the maximum 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 parameter:

`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:

`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:

`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:

`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:

`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:

`property_map` Properties object to initialize SSO with.

Returns

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

`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:

`sso_token_handle`	SSO token handle of session to be invalidated.

Returns

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

`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:

`sso_token_handle` SSO token to check if valid.

Returns

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

`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:

`sso_token_handle` SSO token to refresh.

Returns

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

`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 parameter:

listener The change event listener.

Returns

This function returns am_status_t with one of the following values:

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:

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:

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:

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:

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:

`sso_token_handle` SSO token to validate.

Returns

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

`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.

◆ ◆ ◆ CHAPTER 9

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. 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()" on page 138
- "am_web_cleanup()" on page 138
- "am_web_create_post_page()" on page 139
- "am_web_create_post_preserve_urls()" on page 140
- "am_web_free_memory()" on page 140
- "am_web_get_agent_server_host()" on page 141
- "am_web_get_agent_server_port()" on page 141
- "am_web_get_cookie_name()" on page 142
- "am_web_get_notification_url()" on page 143
- "am_web_get_parameter_value()" on page 143
- "am_web_get_redirect_url()" on page 144
- "am_web_get_token_from_assertion()" on page 145
- "am_web_handle_notification()" on page 146
- "am_web_http_decode()" on page 147
- "am_web_init()" on page 147
- "am_web_is_access_allowed()" on page 148
- "am_web_is_cdsso_enabled()" on page 149
- "am_web_is_debug_on()" on page 150
- "am_web_is_in_not_enforced_ip_list()" on page 150
- "am_web_is_in_not_enforced_list()" on page 151
- "am_web_is_max_debug_on()" on page 151
- "am_web_is_notification()" on page 152
- "am_web_is_postpreserve_enabled()" on page 153
- "am_web_is_valid_fqdn_url()" on page 153
- "am_web_log_always()" on page 154
- "am_web_log_auth()" on page 154
- "am_web_log_debug()" on page 155

- “am_web_log_error()” on page 156
- “am_web_log_info()” on page 156
- “am_web_log_max_debug()” on page 157
- “am_web_log_warning()” on page 157
- “am_web_postcache_data_cleanup()” on page 158
- “am_web_postcache_insert()” on page 158
- “am_web_postcache_lookup()” on page 159
- “am_web_postcache_remove()” on page 160
- “am_web_remove_parameter_from_query()” on page 160

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 parameter:

`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:

`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:

`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:

`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 parameter:

`request_url` The request URL for POST in the HTTP request.

Returns

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

`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 parameter:

`memory` Memory allocated by an `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 value:

`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 value:

`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:

`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:

`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:

`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 parameter value should be freed by the caller using `am_web_free()`.

Returns

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

`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 parameter 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:

`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:

`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:

`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:

`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:

`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:

`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 parameter:

`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:

`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:

`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:

`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:

`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:

`ip` The IP address.

Returns

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

`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:

`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:

`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:

`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:

`request_url` The request URL

Returns

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

`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:

`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:

`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:

`fmt` Formatted string as in `printf`.

Returns

None

am_web_log_auth()

Log the given access allowed or denied message to the Access Manager logs.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_log_auth(am_web_access_t access_type, const char *fmt, ...);
```

Parameters

This function takes the following parameters:

`access_type` `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:

`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:

`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:

`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:

`fmt` Formatted string like in `printf` to be logged.

Returns

None

am_web_log_max_debug()

Log the given message at maximum debug level.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_log_max_debug(const char *fmt, ...);
```

Parameters

This function takes the following parameters:

`fmt` 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:

`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:

`const` `am_web_postcache_data_t` 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:

`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:

`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:

`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:

`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:

`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:

`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:

`AM_SUCCESS` If the call was successful.

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

◆ ◆ ◆ CHAPTER 10

Miscellaneous Functions

This chapter provides a reference to various Sun Java™ System Access Manager 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()" on page 163
- "am_notify()" on page 164
- "am_string_set_allocate()" on page 165
- "am_string_set_destroy()" on page 166
- "am_status_to_name()" on page 166
- "am_status_to_string()" on page 167
- "am_http_cookie_encode()" on page 168
- "am_http_cookie_decode()" on page 168

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.