Sun OpenSSO Enterprise 8.0 C API Reference for Application and Web Policy Agent Developers

Beta

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

Part No: 820–3738–08
October 2008

Copyright 2008 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents or pending patent applications in the U.S. and in other countries.

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, the Solaris logo, the Java Coffee Cup logo, docs.sun.com, Java, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. or its subsidiaries in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Products covered by and information contained in this publication are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical or biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2008 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuelle relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plusieurs brevets américains ou des applications de brevet en attente aux Etats-Unis et dans d'autres pays.

Cette distribution peut comprendre des composants développés par des tierces personnes.

Certaines composants de ce produit peuvent être dérivées du logiciel Berkeley BSD, licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays; elle est licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, le logo Solaris, le logo Java Coffee Cup, docs.sun.com, Java et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc., ou ses filiales, aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui, en outre, se conforment aux licences écrites de Sun.

Les produits qui font l'objet de cette publication et les informations qu'il contient sont régis par la legislation américaine en matière de contrôle des exportations et peuvent être soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes chimiques ou biologiques ou pour le nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers des pays sous embargo des Etats-Unis, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régis par la legislation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement designés, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFACON.

081012@21288

Contents

Preface 13

1 **The C SDK Application Programming Interfaces** 19
Where is the C SDK? 19
What is Included in the C SDK? 20
Header Files 20
Code Samples 21
Developing Web Policy Agents 21
Required C Libraries 22
Solaris Operating System 22
Linux Application Environment 22
Microsoft Windows 23

2 **Authentication Data Types and Functions** 25
The Authentication API for C 25
Authentication Call Sequence 25
Authentication Properties 26
Authentication Data Types 27
`am_auth_context_t` 27
`am_auth_callback_t` 28
`am_auth_locale_t` 29
Authentication Callback Data Types 30
`am_auth_choice_callback_t` 30
`am_auth_confirmation_callback_t` 32
`am_auth_language_callback_t` 33
`am_auth_name_callback_t` 33
`am_auth_password_callback_t` 34

am_auth_text_input_callback_t 35
am_auth_text_output_callback_t 36
Authentication Functions 36
am_auth_abort() 37
am_auth_create_auth_context() 37
am_auth_destroy_auth_context() 38
am_auth_get_callback() 39
am_auth_get_module_instance_names() 40
am_auth_get_organization_name() 41
am_auth_get_sso_token_id() 41
am_auth_get_status() 42
am_auth_has_more_requirements() 43
am_auth_init() 44
am_auth_login() 44
am_auth_logout() 45
am_auth_num_callbacks() 46
am_auth_submit_requirements() 46

3 Policy Data Types and Functions 49
The Policy API for C 49
Resources Strings 50
Resource Traits 50
Policy Evaluation 50
Policy Data Types 51
am_policy_result_t 51
am_policy_t 53
am_resource_traits_t 53
Policy Functions 56
am_policy_compare_urls() 56
am_policy_destroy() 58
am_policy_evaluate() 58
am_policy_evaluate_ignore_url_notenforced() 61
am_policy_get_url_resource_root() 63
am_policy_init() 64
am_policy_invalidate_session() 64

am_policy_is_notification_enabled() 65
am_policy_notify() 66
am_policy_resource_canonicalize() 66
am_policy_resource_has_patterns() 67
am_policy_result_destroy() 67
am_policy_service_init() 68

4 Single Sign-On Data Types and Functions 71
The Single Sign-on API for C 71
Single Sign-on Properties 71
Single Sign-on Calls 73
Non-Web Applications 79
Single Sign-on Data Types 79
am_sso_token_handle_t 79
am_sso_token_listener_func_t 80
Single Sign-on Functions 81
am_sso_add_listener() 81
am_sso_add_sso_token_listener() 83
am_sso_create_sso_token_handle() 85
am_sso_destroy_sso_token_handle() 86
am_sso_get_auth_level() 86
am_sso_get_auth_type() 87
am_sso_get_host() 87
am_sso_get_idle_time 88
am_sso_get_max_idle_time() 88
am_sso_get_max_session_time() 89
am_sso_get_principal() 89
am_sso_get_principal_set() 90
am_sso_get_property() 90
am_sso_get_sso_token_id() 91
am_sso_get_time_left() 91
am_sso_init() 92
am_sso_invalidate_token() 93
am_sso_is_valid_token() 94
am_sso_refresh_token() 94

am_sso_remove_listener() ... 95
am_sso_remove_sso_token_listener() ... 96
am_sso_set_property() ... 97
am_sso_validate_token() ... 98

5 Logging Data Types and Functions ... 101
The Logging API for C ... 101
Logging Data Types ... 102
am_log_record_t ... 102
am_log_module_id_t ... 102
Logging Functions ... 103
am_log_add_module() ... 103
am_log_flush_remote_log() ... 105
am_log_init() ... 105
am_log_is_level_enabled() ... 106
am_log_log() ... 107
am_log_log_record() ... 108
am_log_record_add_loginfo() ... 109
am_log_record_create() ... 110
am_log_record_destroy() ... 111
am_log_record_populate() ... 112
am_log_record_set_log_level() ... 112
am_log_record_set_log_message() ... 113
am_log_record_set_loginfo_props() ... 114
am_log_set_levels_from_string() ... 115
am_log_set_log_file() ... 115
am_log_set_module_level() ... 116
am_log_set_remote_info() ... 117
am_log_vlog() ... 118

6 Mapping Data Types and Functions ... 121
The Mapping API for C ... 121
Mapping Data Types ... 121
am_map_t ... 121
am_map_entry_iter_t ... 122

am_map_value_iter_t 122
Mapping Functions 123
am_map_clear() 123
am_map_copy() 124
am_map_create() 125
am_map_destroy() 125
am_map_entry_iter_destroy() 126
am_map_entry_iter_get_first_value() 126
am_map_entry_iter_get_key() 127
am_map_entry_iter_get_values() 128
am_map_entry_iter_is_entry_valid() 129
am_map_entry_iter_next() 129
am_map_erase() 130
am_map_find() 131
am_map_find_first_value() 131
am_map_for_each() 132
am_map_get_entries() 133
am_map_insert() 134
am_map_size() 135
am_map_value_iter_destroy() 136
am_map_value_iter_get() 136
am_map_value_iter_is_value_valid() 137
am_map_value_iter_next() 137

7 Property Data Types and Functions 139
The Property API for C 139
Property Data Types 139
am_properties_t 139
am_properties_iter_t 140
Property Functions 140
am_properties_copy() 141
am_properties_create() 142
am_properties_destroy() 142
am_properties_get() 143
am_properties_get_boolean() 144

am_properties_get_boolean_with_default() 145
am_properties_get_entries() 145
am_properties_get_positive_number() 146
am_properties_get_signed() 147
am_properties_get_signed_with_default() 148
am_properties_get_unsigned() 148
am_properties_get_unsigned_with_default() 149
am_properties_get_with_default() 150
am_properties_is_set() 151
am_properties_iter_destroy() 152
am_properties_iter_get_key() 152
am_properties_iter_get_value() 153
am_properties_load() 153
am_properties_set() 154
am_properties_store() 155

8 Web Agent Data Types and Functions 157
Web Agent API for C 157
Web Agent Data Types 158
am_web_add_header_in_response_t 158
am_web_free_post_data_t 158
am_web_get_post_data_t 159
am_web_postcache_data_t 160
am_web_render_result_t 160
am_web_request_func_t 161
am_web_request_params_t 161
am_web_set_header_in_request_t 163
am_web_set_method_t 163
am_web_set_user_t 164
post_urls_t 164
Web Agent Function Pointers 165
am_web_add_header_in_response_func_t 165
am_web_free_post_data_func_t 166
am_web_get_cookie_sync_func_t 167
am_web_get_post_data_func_t 168

am_web_render_result_func_t 168
am_web_result_set_header_func_t 169
am_web_result_set_header_attr_in_request_func_t 170
am_web_result_set_header_attr_in_response_func_t 171
am_web_set_header_in_request_func_t 172
am_web_set_method_func_t 173
am_web_set_user_func_t 174
Web Agent Functions 175
am_web_init() for Agents 3.0 176
am_agent_init() 177
am_web_build_advice_response() 177
am_web_check_cookie_in_post() 178
am_web_check_cookie_in_query() 179
am_web_clean_post_urls() 181
am_web_cleanup() 181
am_web_clear_attributes_map() 182
am_web_create_post_page() 182
am_web_create_post_preserve_urls() 183
am_web_delete_agent_configuration() 184
am_web_do_cookie_domain_set() 184
am_web_do_cookies_reset() 185
am_web_free_memory() 186
am_web_get_agent_server_host() 186
am_web_get_agent_configuration() 187
am_web_get_agent_server_port() 187
am_web_get_authType() 188
am_web_get_cookie_name() 188
am_web_get_notification_url() 189
am_web_get_parameter_value() 189
am_web_get_request_url() 190
am_web_get_token_from_assertion() 191
am_web_get_url_to_redirect() 192
am_web_get_user_id_param() 194
am_web_handle_notification() 194
am_web_http_decode() 195
am_web_is_access_allowed() 195

am_web_is_cdsso_enabled() 197
am_web_is_cookie_present() 197
am_web_is_debug_on() 198
am_web_is_in_not_enforced_ip_list() 199
am_web_is_in_not_enforced_list() 200
am_web_is_logout_url() 200
am_web_is_max_debug_on() 201
am_web_is_notification() 202
am_web_is_owa_enabled() 202
am_web_is_owa_enabled_change_protocol() 203
am_web_is_owa_enabled_session_timeout_url() 204
am_web_is_postpreserve_enabled() 204
am_web_is_proxy_override_host_port_set() 205
am_web_is_valid_fqdn_url() 205
am_web_log_always() 206
am_web_log_auth() 207
am_web_log_debug() 207
am_web_log_error() 208
am_web_log_info() 208
am_web_log_max_debug() 209
am_web_log_warning() 209
am_web_logout_cookies_reset() 210
am_web_method_num_to_str() 210
am_web_method_str_to_num() 211
am_web_postcache_data_cleanup() 212
am_web_postcache_insert() 213
am_web_postcache_lookup() 213
am_web_postcache_remove() 214
am_web_process_request() 215
am_web_remove_authnrequest() 216
am_web_remove_parameter_from_query() 217
am_web_result_attr_map_set() 217
am_web_result_num_to_str() 219
am_web_set_cookie() 219

9 Additional Data Types and Functions 221

<am.h> 221

am_cleanup() Syntax 221

am_cleanup() Parameters 222

am_cleanup() Returns 222

<am_notify.h> 222

am_notify() Syntax 222

am_notify() Parameters 222

am_notify() Returns 223

<am_string_set.h> 223

String Data Types 223

String Functions 224

<am_types.h> 225

boolean_t 225

bool_t 225

am_status_t 225

am_status_to_string() 226

<am_utils.h> 228

am_http_cookie_encode() 228

am_http_cookie_decode() 229

Index 231

Preface

The *Sun OpenSSO Enterprise 8.0 C API Reference* provides a listing of application programming interfaces (API) you can use to enable C applications to access the Sun OpenSSO Enterprise components. The *C API Reference* includes function descriptions and syntax.

Who Should Use This Book

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

- C Programming
- Security Assertion Markup Language (SAML) Specifications
- eXtensible Markup Language (XML)
- Lightweight Directory Access Protocol (LDAP)
- HyperText Transfer Protocol (HTTP)
- HyperText Markup Language (HTML)

Related Documentation

Related documentation is available as follows:

- "OpenSSO Enterprise 8.0 Core Documentation" on page 13
- "Related Product Documentation" on page 15

OpenSSO Enterprise 8.0 Core Documentation

The OpenSSO Enterprise 8.0 core documentation set contains the following titles:

- The *Sun OpenSSO Enterprise 8.0 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 OpenSSO Enterprise 8.0 Technical Overview* provides high level explanations of how OpenSSO Enterprise components work together to protect enterprise assets and web-based applications. It also explains basic concepts and terminology.
- The *Sun OpenSSO Enterprise 8.0 Deployment Planning Guide* provides planning and deployment solutions for OpenSSO Enterprise based on the solution life cycle
- The *Deployment Example: Single Sign-On, Load Balancing and Failover Using Sun OpenSSO Enterprise 8.0* provides instructions for building an OpenSSO solution incorporating authentication, authorization and access control. Procedures for load balancing and session failover are also included.
- The *Deployment Example: SAML v2 Using Sun OpenSSO Enterprise 8.0* provides instructions for building an OpenSSO solution incorporating SAML v2 federation. Installation and configuration procedures are included.
- The *Sun OpenSSO Enterprise 8.0 Installation and Configuration Guide* provides information for installing and configuring OpenSSO Enterprise.
- The *Sun OpenSSO Enterprise 8.0 Performance Tuning and Troubleshooting Guide* provides information on how to tune OpenSSO Enterprise and its related components for optimal performance.
- The *Sun OpenSSO Enterprise 8.0 Administration Guide* describes administrative tasks such as *how to create a realm* and *how to configure a policy*. Most of the tasks described can be performed using the administration console as well as the `ssoadm` command line utilities.
- The *Sun OpenSSO Enterprise 8.0 Administration Reference* is a guide containing information about the command line interfaces, configuration attributes, internal files, and error codes. This information is specifically formatted for easy searching.
- The *Sun OpenSSO Enterprise 8.0 Developer's Guide* offers information on how to customize OpenSSO Enterprise 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 OpenSSO Enterprise 8.0 C API Reference for Application and Web Policy Agent Developers* (this guide) provides summaries of data types, structures, and functions that make up the public OpenSSO Enterprise C SDK for application and web agent development.
- The *Sun OpenSSO Enterprise 8.0 Java API Reference* provides information about the implementation of Java packages in OpenSSO Enterprise.
- The *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for Web Agents* and *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for J2EE Agents* provide an overview of the policy functionality and policy agents available for OpenSSO Enterprise.

Updates to the *Release Notes* and links to modifications of the core documentation can be found on the OpenSSO Enterprise page at docs.sun.com. Updated documents will be marked with a revision date.

Related Product Documentation

The following table provides links to documentation for related products.

TABLE P–1 Related Product Documentation

Product	Link
Sun Java System Directory Server 6.3	http://docs.sun.com/coll/1224.4
Sun Java System Web Server 7.0 Update 3	http://docs.sun.com/coll/1653.3
Sun Java System Application Server 9.1	http://docs.sun.com/coll/1343.4
Sun Java System Message Queue 4.1	http://docs.sun.com/coll/1307.3
Sun Java System Web Proxy Server 4.0.6	http://docs.sun.com/coll/1311.6
Sun Java System Identity Manager 7.1	http://docs.sun.com/coll/1514.3

Searching Sun Product Documentation

Besides searching Sun product documentation from the docs.sun.com[SM] web site, you can use a search engine by typing the following syntax in the search field:

search-term `site:docs.sun.com`

For example, to search for "broker," type the following:

```
broker site:docs.sun.com
```

To include other Sun web sites in your search (for example, java.sun.com, www.sun.com, and developers.sun.com), use `sun.com` in place of `docs.sun.com` in the search field.

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (http://www.sun.com/documentation/)
- Support (http://www.sun.com/support/)
- Training (http://www.sun.com/training/)

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.

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 full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the title of this book is *Sun OpenSSO Enterprise 8.0 Technical Overview*, and the part number is 820–3740.

Typographical Conventions

The following table describes the typographic conventions that are used in this deployment example.

TABLE P–2 Typographic Conventions

Typeface	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*.

TABLE P–2 Typographic Conventions *(Continued)*

Typeface	Meaning	Example
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

CHAPTER 1

The C SDK Application Programming Interfaces

Sun OpenSSO Enterprise provides C application programming interfaces (API) and related information to form a software development kit (SDK) that can be used to enable external C applications to participate in OpenSSO Enterprise authentication, authorization, single sign-on (SSO), and logging operations. Additionally, the C SDK is a development kit for web agents — allowing developers to create web agents for containers that have not yet been provided for OpenSSO Enterprise. This chapter covers the following topics:

- "Where is the C SDK?" on page 19
- "What is Included in the C SDK?" on page 20
- "Developing Web Policy Agents" on page 21
- "Required C Libraries" on page 22

Where is the C SDK?

The OpenSSO Enterprise C SDK can be found in the `opensso` directory created on the machine to which OpenSSO Enterprise is deployed. From inside the top-level `opensso` directory, change into the `libraries/native/agent-csdk` directory where you will find `agent-csdk.zip` and a `README`. Within `agent-csdk.zip` are the following 32–bit and 64–bit libraries:

- `agent-csdk-solaris-sparc-32.tar.gz` is the C SDK for the Solaris SPARC 32–bit platform.
- `agent-csdk-solaris-sparc-64.tar.gz` is the C SDK for the Solaris SPARC 64–bit platform.
- `agent-csdk-solaris-x86.tar.gz` is the C SDK for the Solaris x86 32–bit platform.
- `agent-csdk-solaris-x86.tar.gz` is the C SDK for the Solaris x86 64–bit platform.
- `agent-csdk-linux-32.tar.gz` is the C SDK for the Linux 32–bit platform.
- `agent-csdk-linux-64.tar.gz` is the C SDK for the Linux 64–bit platform.
- `agent-csdk-windows-32.zip` is the C SDK for the Windows 32–bit platform.
- `agent-csdk-windows-64.zip` is the C SDK for the Windows 64–bit platform.

What is Included in the C SDK?

Each library contains header files, samples, and README files containing information on how to implement the C SDK. More information is in the following sections.

- "Header Files" on page 20
- "Code Samples" on page 21

Header Files

A *header file* is a text file that contains pieces of code written in the C programming language. The name of a header file, by convention, ends with the `.h` extension. It is inserted inside a program by coding the `#include` preprocessor directive. The OpenSSO Enterprise C header files are:

`<am.h>`	General utility routines provided by the OpenSSO Enterprise library.
`<am_auth.h>`	Data types and functions for developing custom authentication modules.
`<am_log.h>`	Data types and functions for logging on the local system or the OpenSSO Enterprise host.
`<am_map.h>`	Data types and functions for creating, destroying, and manipulating the map objects used by OpenSSO Enterprise.
`<am_notify.h>`	Data types and functions for implementing notifications.
`<am_policy.h>`	Data types and functions for using OpenSSO Enterprise policy objects.
`<am_properties.h>`	Data types and functions for property maps used by clients of the OpenSSO Enterprise client API.
`<am_sso.h>`	Data types and functions for implementing SSO.
`<am_string_set.h>`	Data types and functions for manipulating strings.
`<am_types.h>`	Common types and macros provided by OpenSSO Enterprise.
`<am_utils.h>`	Functions to encode/decode HTTP cookies.
`<am_web.h>`	Data types and functions intended for use by OpenSSO Enterprise web agents.

Code Samples

OpenSSO Enterprise provides code samples that demonstrate how you can use the API to connect C applications to the OpenSSO Enterprise framework. The code samples are:

`am_auth_test.c`	Demonstrates the basic usage of the authentication API used to login to an instance of OpenSSO Enterprise.
`am_log_test.c`	Demonstrates the basic usage of the logging API used to write a message to the OpenSSO Enterprise logs.
`am_policy_test.c`	Demonstrates the basic usage of the policy API to evaluate access for specified resources.
`am_sso_test.c`	Demonstrates the basic usage of the SSO API to perform session operations.
`apache_agent.c`	Demonstrates how to use the policy API to build a web agent for the Apache Web Server.

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

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

Note – `am_web_agent_test.c`, referred to in this file, is no longer used.

Developing Web Policy Agents

The C SDK can be used to develop web policy agents. Web agents that are currently available for OpenSSO Enterprise can be downloaded from the agents download page. Web agents that are not yet available can be developed using the C API and the included sample source files as a blueprint. See `README.TXT` in the `samples` directory for more information on using this *web policy agent development toolkit*.

Required C Libraries

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

- "Solaris Operating System" on page 22
- "Linux Application Environment" on page 22
- "Microsoft Windows" on page 23

Solaris Operating System

For the Solaris™ operating System, set the `LD_LIBRARY_PATH` environment variable to include:

```
/usr/lib/mps:opensso-base/is_csdk/lib:/usr/lib:/usr/ucblib
```

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

These directories contain the following shared libraries:

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

Linux Application Environment

For the Linux application environment, set the `LD_LIBRARY_PATH` environment variable to include:

```
/usr/lib/mps:opensso-base/is_csdk/lib:/usr/lib:/usr/ucblib
```

This directory contains the following shared libraries:

- `libamsdk.so`
- `libxml2.so`
- `libssl3.so`

- `libnss3.so`
- `libplc4.so`
- `libplds4.so`
- `libnspr4.so`

Microsoft Windows

When using Microsoft® Windows, you must have the `\\`*opensso-base*`\is_csdk\bin` directory in your path before launching the samples. Alternatively, you can run the `run.bat` script to launch the samples. The script will set your path appropriately.

CHAPTER 2

Authentication Data Types and Functions

Sun OpenSSO Enterprise contains public data types and functions you can use in developing custom authentication modules. This chapter provides information and a reference guide to the authentication application programming interfaces (API). Reference summaries include a short description, syntax, parameters and returns. Prototypes are contained in the `<am_auth.h>` header file located in the `opensso/libraries/native/agent-csdk/is_csdk/include` directory (when `opensso.war` is exploded on Solaris/SPARC). The sample source `am_auth_test.c` demonstrates the basic usage of the API to login to an instance of OpenSSO Enterprise. This chapter contains the following sections:

- "The Authentication API for C" on page 25
- "Authentication Data Types" on page 27
- "Authentication Callback Data Types" on page 30
- "Authentication Functions" on page 36

The Authentication API for C

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

Authentication Call Sequence

The sequence of calls necessary to authenticate to OpenSSO Enterprise begins with the function call `am_auth_create_auth_context()`. This call returns an `am_auth_context` structure that is then used for the rest of the authentication calls. Once the structure has been initialized, the

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

Note – The `remote-auth.dtd` is the template used to format XML authentication requests sent to OpenSSO Enterprise and to parse XML authentication responses received by the external application. The attributes in the requests/responses correspond to elements in the `remote-auth.dtd`. In the example, user name corresponds to the `NameCallback` element and password to the `PasswordCallback` element in the `remote-auth.dtd`. More information on `remote-auth.dtd` can be found in Chapter 2, "Using the Authentication Interfaces," in *Sun OpenSSO Enterprise 8.0 Developer's Guide*.

Authentication Properties

With the newly developed policy agents 3.0, `AMAgent.properties` has been replaced with `OpenSSOAgentBootstrap.properties` and `OpenSSOAgentConfiguration.properties`. Properties in `OpenSSOAgentBootstrap.properties` are mandatory for any C API to work. Properties in `OpenSSOAgentConfiguration.properties` will only be used if the repository type of the agent user is `local`. If the repository type is `centralized`, any required properties not in `OpenSSOAgentBootstrap.properties` will be retrieved from the OpenSSO Enterprise server.

Note – See "Centralized Agent Configuration" in *Sun OpenSSO Enterprise 8.0 Technical Overview* for more information.

The following table lists the mandatory properties in `OpenSSOAgentBootstrap.properties`.

TABLE 2–1 Policy Agent 3.0 Properties Needed by the Authentication API for C

Property	Definition
`com.sun.identity.agents.config.naming.url`	URL of the OpenSSO Enterprise Naming Service in the format: **`http://`***server*`.`*domain*`:`*port*`/`*URI*`/`**`namingservice`**
`com.sun.identity.agents.config.local.logfile`	The logging directory in the format: *path-to-directory*`/`**`logs/auth-log`**
`com.sun.identity.agents.config.debug.level`	The level at which logs are written in the format: **`all:`**`#` where # is the level 5 being the highest, 3 medium and 1 the lowest.
`com.sun.identity.agents.config.sslcert.dir`	Path to the directory containing the certificate and key databases for Secure Sockets Layer (SSL).
`com.sun.identity.agents.config.certdb.prefix`	Set this property if the certificate databases in the directory specified by **`com.sun.identity.agents.config.sslcert.dir`** has a prefix.
`com.sun.identity.agents.config.certdb.password`	The password to the `key3.db` file. **Note –** This property may be added to `OpenSSOAgentBootstrap.properties`.
`com.sun.identity.agents.config.trust.server.certs`	Defines whether or not to trust SSL certificates not defined in the certificate database. Takes a value of `true` or `false` where `true` enables trust.
`com.sun.identity.agents.config.certificate.alias`	The nickname of the client certificate in the `cert7.db`. **Note –** This property may be added to `OpenSSOAgentBootstrap.properties`.

Authentication Data Types

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

- "`am_auth_context_t`" on page 27
- "`am_auth_callback_t`" on page 28
- "`am_auth_locale_t`" on page 29

`am_auth_context_t`

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

Syntax

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

Members

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

Memory Concerns

The implementation takes care of creating memory.

am_auth_callback_t

Primary callback type for authentication.

Details

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

Syntax

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

Members

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

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

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

`callback_info` Represents the defined `callback_type`. More information on the specific callbacks can be found in "Authentication Callback Data Types" on page 30.

Memory Concerns

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

am_auth_locale_t

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

Details

For more information, see "`am_auth_language_callback_t`" on page 33.

Syntax

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

Members

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

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

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

Authentication Callback Data Types

This section contains further details on the callback types as discussed in "`am_auth_callback_t`" on page 28. They are:

- "`am_auth_choice_callback_t`" on page 30
- "`am_auth_confirmation_callback_t`" on page 32
- "`am_auth_language_callback_t`" on page 33
- "`am_auth_name_callback_t`" on page 33
- "`am_auth_password_callback_t`" on page 34
- "`am_auth_text_input_callback_t`" on page 35
- "`am_auth_text_output_callback_t`" on page 36

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

---

am_auth_choice_callback_t

Displays a list of choices and submits the selection back to the OpenSSO Enterprise Authentication Service.

Details

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

Syntax

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

Members

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

Memory Concerns

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

am_auth_confirmation_callback_t

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

Details

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

Syntax

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

Members

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

Memory Concerns

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

am_auth_language_callback_t

Retrieves the locale for localizing text.

Details

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

Note – See "am_auth_locale_t" on page 29 for the individual components.

Syntax

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

Members

locale	Pointer to the am_auth_locale_t object defining the locale.
response	Pointer to the am_auth_locale_t object being submitted back to OpenSSO Enterprise.

Memory Concerns

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

am_auth_name_callback_t

Retrieves user name information.

Details

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

Syntax

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

Members

`prompt`	Pointer to the user's prompt.
`default_name`	Pointer to the default name displayed with the user prompt, if any.
`response`	Pointer to the name submitted back to OpenSSO Enterprise.

Memory Concerns

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

am_auth_password_callback_t

Retrieves user password information.

Details

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

Syntax

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

Members

`prompt`	Pointer to the user's prompt.

`echo_on`	Takes a value based on the `boolean_t` defined in the `<am_types.h>` header file. Set to `B_TRUE` to display the password as it is typed.
`response`	Pointer to the password submitted back to OpenSSO Enterprise.

Memory Concerns

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

am_auth_text_input_callback_t

Retrieves generic textual information.

Details

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

Syntax

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

Members

`prompt`	Pointer to the user's prompt.
`default_text`	Pointer to the default text to be displayed with the user prompt.
`response`	Pointer to the text submitted back to OpenSSO Enterprise.

Memory Concerns

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

am_auth_text_output_callback_t

Displays information messages, warning messages, and error messages.

Details

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

Syntax

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

Members

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

Memory Concerns

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

Authentication Functions

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

- "am_auth_abort()" on page 37
- "am_auth_create_auth_context()" on page 37
- "am_auth_destroy_auth_context()" on page 38
- "am_auth_get_callback()" on page 39
- "am_auth_get_module_instance_names()" on page 40
- "am_auth_get_organization_name()" on page 41
- "am_auth_get_sso_token_id()" on page 41
- "am_auth_get_status()" on page 42
- "am_auth_has_more_requirements()" on page 43
- "am_auth_init()" on page 44
- "am_auth_login()" on page 44
- "am_auth_logout()" on page 45

- "am_auth_num_callbacks()" on page 46
- "am_auth_submit_requirements()" on page 46

am_auth_abort()

Aborts an authentication process that has not been completed.

Syntax

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

Parameters

This function takes the following parameter:

`auth_ctx` The `am_auth_context_t` type.

Note – See "am_auth_context_t" on page 27 for information.

Returns

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

`AM_SUCCESS` If the process was successfully stopped.

`AM_INVALID_ARGUMENT` If the `auth_ctx` parameter is `NULL`.

am_auth_create_auth_context()

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

Syntax

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

Parameters

This function takes the following parameters:

`auth_ctx` Pointer to the `am_auth_context_t` type.

Note – See "`am_auth_context_t`" on page 27 for information.

`org_name` Pointer to the name of the organization for which the authentication context is being initialized. May be `NULL` to use the value defined in the agent configuration properties.

`cert_nick_name` Pointer to the alias of the certificate being used if the application will connect securely to OpenSSO Enterprise. May be `NULL` if the connection is not secure.

`url` Pointer to the OpenSSO Enterprise Naming Service URL. May be `NULL` to use the Naming Service URL defined in the agent configuration properties.

Returns

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

`AM_SUCCESS` If the authentication context was successfully created.

`AM_NO_MEMORY` If unable to allocate memory for the handle.

`AM_INVALID_ARGUMENT` If the `auth_ctx` parameter is `NULL`.

`AM_AUTH_CTX_INIT_FAILURE` If the authentication initialization failed.

am_auth_destroy_auth_context()

Eliminates the specified authentication context.

Syntax

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

Parameters

This function takes the following parameter:

`auth_ctx` The `am_auth_context_t` type.

Note – See "`am_auth_context_t` on page 27" for information.

Returns

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

`AM_SUCCESS` If the pointer was successfully destroyed.

`AM_INVALID_ARGUMENT` If the `auth_ctx` parameter is `NULL`.

am_auth_get_callback()

Retrieves the appropriate callback structure to populate with authentication requirements.

Syntax

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

Parameters

This function takes the following parameters:

`auth_ctx` The `am_auth_context_t` type.

Note – See "`am_auth_context_t` on page 27" for information.

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

Returns

This function returns a pointer to the `am_auth_callback_t` type. See "`am_auth_callback_t` on page 28" for more information.

am_auth_get_module_instance_names()

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

Details

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

Syntax

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

Parameters

This function takes the following parameters:

`auth_ctx`	The `am_auth_context_t` type.
	Note – See "`am_auth_context_t` on page 27" for information.
`module_inst_names_ptr`	Pointer to a pointer to the `am_string_set_t` type.

Returns

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

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

Memory Concerns

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

am_auth_get_organization_name()

Retrieves the organization to which the user is authenticated.

Syntax

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

Parameters

This function takes the following parameter:

`auth_ctx` The `am_auth_context_t` type.

Note – See "am_auth_context_t" on page 27 for information.

Returns

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

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

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

am_auth_get_sso_token_id()

Retrieves the session identifier for the authenticated user.

Details

The *SSOTokenID* is a randomly-generated string that represents an authenticated user. See "Single Sign-on Token Handles" on page 75 for more information.

Syntax

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

Parameters

This function takes the following parameter:

`auth_ctx` The `am_auth_context_t` type.

Note – See "`am_auth_context_t` on page 27" for information.

Returns

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

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

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

`am_auth_get_status()`

Retrieves the state of the authentication process.

Syntax

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

Parameters

This function takes the following parameter:

`auth_ctx` The `am_auth_context_t` type.

Note – See "`am_auth_context_t` on page 27" for information.

Returns

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

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

`AM_AUTH_STATUS_FAILED` The login process has failed.

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

am_auth_has_more_requirements()

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

Details

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

Syntax

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

Parameters

This function takes the following parameter:

auth_ctx The am_auth_context_t type.

Note – See "am_auth_context_t" on page 27 for information.

Returns

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

B_TRUE If there are more requirements.

B_FALSE If there are no more requirements.

am_auth_init()

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

Syntax

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

Parameters

This function takes the following parameter:

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

Note – See "`am_properties_t`" on page 139 for information.

Returns

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

`AM_SUCCESS`	If the initialization of the library is successful.
`AM_NO_MEMORY`	If unable to allocate memory during initialization.
`AM_INVALID_ARGUMENT`	If `auth_init_params` is `NULL`.
Others	See "`am_status_t`" on page 225.

am_auth_login()

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

Syntax

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

Parameters

This function takes the following parameters:

`auth_ctx` The `am_auth_context_t` type.

Note – See "`am_auth_context_t`" on page 27 for information.

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

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

`value` Pointer to the authentication module being used.

Note – See "Authentication Modules" in *Sun OpenSSO Enterprise 8.0 Administration Guide* for more information.

Returns

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

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

am_auth_logout()

Logs out the user.

Syntax

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

Parameters

This function takes the following parameter:

`auth_ctx` The `am_auth_context_t` type.

Note – See "`am_auth_context_t` on page 27" for information.

Returns

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

`AM_SUCCESS` If the logout process was successfully completed.

`AM_INVALID_ARGUMENT` If the `auth_ctx` parameter is `NULL`.

am_auth_num_callbacks()

Retrieves the number of callbacks.

Syntax

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

Parameters

This function takes the following parameters:

`auth_ctx` The `am_auth_context_t` type.

Note – See "`am_auth_context_t` on page 27" for information.

Returns

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

am_auth_submit_requirements()

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

Syntax

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

Parameters

This function takes the following parameter:

`auth_ctx` The `am_auth_context_t` type.

Note – See "`am_auth_context_t` on page 27" for information.

Returns

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

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

CHAPTER 3

Policy Data Types and Functions

Sun OpenSSO Enterprise contains public data types and functions you can use to communicate with the Policy Service. Reference summaries include a short description, syntax, parameters and returns. The code is contained in the `<am_policy.h>` header file. The sample source `am_policy_test.c` demonstrates the basic usage of the policy API. This chapter contains the following sections:

- "The Policy API for C" on page 49
- "Policy Data Types" on page 51
- "Policy Functions" on page 56

The Policy API for C

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

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

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

Resources Strings

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

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

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

Resource Traits

The set of characteristics needed to define a resource is called a *resource trait*. Resource traits are taken as a parameter during service initialization in the "`am_resource_traits_t`" on page 53. Using the resource traits, the Policy Service constructs a resource graph for policy evaluation in which the relation between all resources in the system spans out like a tree from the root of the given resource. Thus, the service developer must provide the means to extract the root of the given resource. In a URL, the *protocol*`://`*resource-host*`.`*domain*`:`*port* portion represents the root.

Policy Evaluation

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

- `am_map_t` provides a key to multiple key/value mapping. If the evaluation requires certain environment parameters like the IP address of the requester, it may be passed using this structure. See "`am_map_t`" on page 121 for more information.
- `am_properties_t` provides a key to single value mapping. `am_properties_t` provides the additional functionality of loading a configuration file and getting values of specific data types. See "`am_properties_t`" on page 139 for more information.

Policy Data Types

The types defined in <am_policy.h> are:

- "am_policy_result_t" on page 51
- "am_policy_t" on page 53
- "am_resource_traits_t" on page 53

am_policy_result_t

Carries the evaluation results from the Policy Service.

Details

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

Syntax

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

Members

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

Note – For information on advices, see "Policy Advices" in *Sun OpenSSO Enterprise 8.0 Administration Guide*.

`attr_profile_map`

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

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

`attr_session_map`

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

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

`attr_response_map`

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

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

`advice_string`

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

Memory Concerns

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

am_policy_t

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

Syntax

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

Members

am_policy_t has no members.

am_resource_traits_t

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

Syntax

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

Members

cmp_func_ptr

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

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

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

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

`rsrc_traits` Pointer to the resource traits structure containing data regarding a policy.

`policy_res_name` Pointer to the name of the resource being protected.

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

`use_patterns` Based on the `boolean_t` defined in the `<am_types.h>` header file, `B_TRUE` indicates that the function will use or recognize patterns when comparing resources.

`has_patterns`

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

`B_TRUE` If `resource_name` has patterns.

`B_FALSE` Otherwise.

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

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

`get_resource_root`

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

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

`B_FALSE` Otherwise.

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

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

`root_resource_name` Buffer to hold the resource root.

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

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

`B_TRUE` Ignore case for all functions in this structure.

`B_FALSE` Otherwise.

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

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

`resource` Pointer to a resource name. This could be the resource being requested or the resource defined in the policy.

`c_resource` Output of the canonical resource name.

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

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

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

`resource_str` Pointer to the string returned in the `canonicalize` function.

Policy Functions

The functions defined in <am_policy.h> are:

- "am_policy_compare_urls()" on page 56
- "am_policy_destroy()" on page 58
- "am_policy_evaluate()" on page 58
- "am_policy_evaluate_ignore_url_notenforced()" on page 61
- "am_policy_get_url_resource_root()" on page 63
- "am_policy_init()" on page 64
- "am_policy_invalidate_session()" on page 64
- "am_policy_is_notification_enabled()" on page 65
- "am_policy_notify()" on page 66
- "am_policy_resource_canonicalize()" on page 66
- "am_policy_resource_has_patterns()" on page 67
- "am_policy_result_destroy()" on page 67
- "am_policy_service_init()" on page 68

Note – Before invoking any of the policy functions, am_web_init() should be invoked by passing the bootstrap property and the local configuration file as parameters, and am_agent_initialized() should be invoked by passing the address of the boolean variable as a parameter — implying that am_web.h should be included in the file.

---

am_policy_compare_urls()

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

Syntax

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

Parameters

This function takes the following parameter:

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

Note – See "am_resource_traits_t" on page 53 for more information.

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

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

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

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

Returns

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

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

`AM_EXACT_MATCH` If both URLs match exactly as in, for example, if the URL for the resource is `http://example.sun.com:90/index.html` and the URL in the policy is `http://example.sun.com:90/index.html`.

`AM_EXACT_PATTERN_MATCH` This result is returned if the URL to which the policy applies matches the URL to which access is requested as in, for example, if the URL for the resource is `http://example.sun.com:90/index.html` and the URL in the policy is `http://example.sun.com:90/*`. Distinction is not made between an `EXACT_MATCH` or a pattern match.

`AM_NO_MATCH` If the URLs do not match.

`AM_SUB_RESOURCE_MATCH`	If the requested URL is found to be a sub-resource of the URL defined in the policy.
`AM_SUPER_RESOURCE_MATCH`	If the requested URL is found to be a parent of the URL defined in the policy.

am_policy_destroy()

Destroys an initialized instance of a policy evaluator object.

Details

An instance is initialized for each policy request.

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

Syntax

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

Parameters

This function takes the following parameter:

`policy`	Integer specifying the object being destroyed.

Returns

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

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

am_policy_evaluate()

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

Note – `am_policy_evaluate()` has been deprecated. See "`am_policy_evaluate_ignore_url_notenforced()`" on page 61.

Details

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

Syntax

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

Parameters

This function takes the following parameters:

`policy_handle` Integer specifying the object being evaluated.

`sso_token` Pointer to the session token (`SSOTokenID`) of the authenticated user.

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

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

`action_name` Pointer to the action requested.

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

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

Note – See "`am_map_t`" on page 121 for more information.

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

Note – This must be enabled in the agent configuration properties. See "`am_policy_result_t`" on page 51 for information on how this is done. See "`am_map_t`" on page 121 for more information on map objects.

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

Note – See "`am_policy_result_t`" on page 51 for more information.

Returns

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

`AM_SUCCESS` If the call was successful.

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

Memory Concerns

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

am_policy_evaluate_ignore_url_notenforced()

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

Details

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

Note – See *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for Web Agents*, or *Sun Java System Access Manager Policy Agent 2.2 User's Guide* for more information.

Syntax

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

Parameters

This function takes the following parameters:

policy_handle — Integer specifying the object being evaluated.

sso_token — Pointer to the session token (SSOTokenID) of the authenticated user.

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

---

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

`action_name`	Pointer to the action requested. **Note –** An *action* is the operation to be performed on the resource. Web server actions are `POST` and `GET`. An allowable action for a human resources service , for example, can change a home telephone number.
`env_parameter_map`	Map object which contains environment variables (IP address, host name, etc.) used for evaluation by the Policy Service. **Note –** See "`am_map_t`" on page 121 for more information.
`policy_response_map_ptr`	Pointer to a map object which contains all the profile, session and response attributes fetched. **Note –** This must be enabled in the agent configuration properties. See "`am_policy_result_t`" on page 51 for information on how this is done. See "`am_map_t`" on page 121 for more information on map objects.
`policy_result`	Pointer to the `am_policy_result_t` type to store the result. **Note –** See "`am_policy_result_t`" on page 51 for more information.
`ignorePolicyResult`	Based on the `am_bool_t` defined in the `<am_types.h>` header file, `AM_TRUE` indicates that policy evaluation will not be done for the URL.

Returns

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

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

Memory Concerns

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

am_policy_get_url_resource_root()

Extracts the root of a given URL.

Details

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

Syntax

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

Parameters

This function takes the following parameters:

`resource_name`	Pointer to the protected resource URL.
`resource_root`	Pointer to the location where the resource root will be written.
`length`	Value based on the `size_t` defined in the standard `<stddef.h>` header file that reflects the size of the `resource_root` buffer.

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

Returns

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

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

am_policy_init()

Initializes the OpenSSO Enterprise Policy Service.

Syntax

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

Parameters

This function takes the following parameter:

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

Note – See "`am_properties_t`" on page 139 for more information.

Returns

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

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

Memory Concerns

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

am_policy_invalidate_session()

Cancels the specified session.

Syntax

```
#include "am_policy.h"
AM_EXPORT am_status_t
```

```
am_policy_invalidate_session(am_policy_t policy_handle,
                             const char *ssoTokenId);
```

Parameters

This function takes the following parameters:

`policy_handle` Integer specifying the object being evaluated.

`ssoTokenId` Pointer to the session token of the authentication user.

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

Returns

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

`AM_SUCCESS` If the call was successful.

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

am_policy_is_notification_enabled()

Checks whether the notification functionality is enabled.

Syntax

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

Parameters

This function takes the following parameter:

`policy_handle` Integer specifying the object being evaluated.

Returns

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

`0` If notification is disabled.

`non-zero` If notification is enabled.

am_policy_notify()

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

Syntax

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

Parameters

This function takes the following parameters:

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

Returns

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

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

am_policy_resource_canonicalize()

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

Syntax

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

Parameters

This function takes the following parameters:

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

Returns

This function does not return a value.

am_policy_resource_has_patterns()

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

Syntax

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

Parameters

This function takes the following parameter:

`resource_name`	Pointer to the resource being evaluated.

Returns

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

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

am_policy_result_destroy()

Destroys the specified `am_policy_result_t` structure type.

Note – See "`am_policy_result_t`" on page 51 for more information.

Syntax

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

Parameters

This function takes the following parameter:

`result` Pointer to the `am_policy_result_t` structure type being destroyed.

Returns

This function does not return a value.

am_policy_service_init()

Initializes one instance of the OpenSSO Enterprise Policy Service for policy evaluation.

Syntax

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

Parameters

This function takes the following parameters:

`service_name` Pointer to the name for the Policy Service.

`instance_name` Pointer to the name of the instance being initialized.

`rsrc_traits` Pointer to a `am_resource_traits_t` structure type.

Note – See "`am_resource_traits_t`" on page 53 for more information.

`service_config_properties` Pointer to the properties used to initialize the Policy Service instance.

Note – See "`am_properties_t`" on page 139 for more information.

`policy_handle_ptr`	Pointer to the integer specifying the object being evaluated.

Returns

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

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

CHAPTER 4

Single Sign-On Data Types and Functions

Sun OpenSSO Enterprise contains public data types and functions you can use to communicate with the Session Service for single sign-on. Reference summaries include a short description, syntax, parameters and returns. The code is contained in the `<am_sso.h>` header file. The sample source `am_sso_test.c` demonstrates the basic usage of the single sign-on API. This chapter contains the following sections:

- "The Single Sign-on API for C" on page 71
- "Single Sign-on Data Types" on page 79
- "Single Sign-on Functions" on page 81

The Single Sign-on API for C

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

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

Single Sign-on Properties

The properties required for SSO are not read from AMAgent.properties but are picked up from either the OpenSSO server (if the mode is centralized) or from the local configuration file (if the mode is local). P

Certain properties must be read and passed to `am_sso_init()` in order to initialize the Session Service. Thus, `am_sso_init()` must be called before any other single sign-on interface. The

properties required for single sign-on are retrieved from OpenSSO Enterprise if using centralized agent configuration, and from the configuration file local to the agent if using local agent configuration.

Note – See *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for Web Agents* or *Sun Java System Access Manager Policy Agent 2.2 User's Guide* for more information.

Before using the API be sure the properties in the following table are set.

TABLE 4–1 Single Sign-on Properties Defined During Web Policy Agent Installation

Property	Definition
`com.sun.am.naming.url`	Specifies the URL for the Naming Service which, in turn, finds the URL of the Session Service. This property must be set as: **`com.sun.am.naming.url =`** *protocol*`://`*OpenSSO Enterprise-host*`.`*domain*`:`*port*`/`*URI*`/`**`namingservice`**
`com.sun.am.notification.enable`	Specifies whether the Notification Service will be used to update the cache. If enabled, a URL where notification messages from OpenSSO Enterprise are sent must be specified. This property is set as: **`com.sun.am.notification.enable =`** *true* | *false* **Note –** If `com.sun.am.notification.enable` is not found in the properties file, the default value is `false`.
`com.sun.am.notification.url`	If `com.sun.am.notification.enable` is set to true, the value of this property specifies a URL where notification messages from OpenSSO Enterprise are sent. This property is set as: **`com.sun.am.notification.url =`** *protocol*`//`*host*`.`*domain*`:`*port*`/`*notification_URL*
`com.sun.am.sso.polling.period`	Specifies how often, in minutes, the cache should be checked for entries that have reached the cache entry life time. This property must be set as: **`com.sun.am.sso.checkCacheInterval=`***#* **Note –** By default, this property is not part of the agent configuration properties but can be added when needed.

TABLE 4–1 Single Sign-on Properties Defined During Web Policy Agent Installation *(Continued)*

Property	Definition
`com.sun.am.sso.max_threads`	Specifies the maximum number of threads the single sign-on API for C should invoke for handling notifications. The API maintains a thread pool and invokes a thread for each notification. If the maximum number of threads has been reached, the notification will wait until a thread is available. This property must be set as: **`com.sun.am.sso.maxThreads=`**# If not specified the default maximum number of threads is 10. **Note –** By default, this property is not part of the agent configuration properties but can be added when needed.

For more information, see the *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for Web Agents* or *Sun Java System Access Manager Policy Agent 2.2 User's Guide*.

Single Sign-on Calls

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

- "Initialization and Cleanup" on page 73
- "Single Sign-on Token Handles" on page 75
- "Retrieving and Setting Properties" on page 76
- "Listening and Notification" on page 78

Initialization and Cleanup

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

Note – For more information on `am_cleanup()`, see Chapter 9, "Additional Data Types and Functions."

```
#include <am_sso.h>

boolean_t agentInitialized = B_FALSE;
const char* prop_file = "../../config/OpenSSOAgentBootstrap.properties";
const char* config_file = "../../config/OpenSSOConfiguration.properties";

am_web_init (prop_file, config_file);
am_agent_init(&agentinitialized)
```

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

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

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

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

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

   /* do sso functions
    * ...
    */

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

    /* exit program successfully. */
```

```
        return 0;
    }
```

Single Sign-on Token Handles

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

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

- "Creating Single Sign-on Token Handles" on page 75
- "Validating Single Sign-on Token Handles" on page 75
- "Destroying Session Token Handles" on page 75

Creating Single Sign-on Token Handles

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

Validating Single Sign-on Token Handles

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

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

Destroying Session Token Handles

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

Tip – Although this ends the session for the user, the proper way to log out is by using `am_auth_logout()` as described in "`am_auth_logout()`" on page 45. Not using `am_auth_logout()` will result in authentication resources associated with the session remaining on the server unnecessarily until the session has timed out.

Retrieving and Setting Properties

The following code sample shows how you might use the `am_sso_get_property()` and `am_sso_set_property()` functions. For additional information, see "`am_sso_get_property()`" on page 90 and "`am_sso_set_property()`" on page 97.

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

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

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

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

     /* check if session is valid using validate. This also updates the handle with
         /*info from the server */
     status = am_sso_validate_token(sso_handle);
     if (status == AM_SUCCESS) {
         printf("Session state is valid.\\n");
     } else if (status == AM_INVALID_SESSION) {
         printf("Session status is invalid.\\n");
     } else {
         printf("Error validating sso token.\\n");
         return 1;
     }

     /* get info on the session */
     printf("SSO Token ID is %s.\\n", am_sso_get_sso_token_id(sso_handle));
     printf("Auth type is %s.\\n", am_sso_get_auth_type(sso_handle));
     printf("Auth level is %d.\\n", am_sso_get_auth_level(sso_handle));
```

```
printf("Idle time is %d.\\n", am_sso_get_idle_time(sso_handle));
printf("Max Idle time is %d.\\n", am_sso_get_max_idle_time(sso_handle));
printf("Time left is %d.\\n", am_sso_get_time_left(sso_handle));
printf("Max session time is %d.\\n", am_sso_get_max_session_time(sso_handle));
printf("Principal is %s.\\n", am_sso_get_principal(sso_handle));
printf("Host is %s.\\n", am_sso_get_host(sso_handle));
principal_set = am_sso_get_principal_set(sso_handle);
if (principal_set == NULL) {
       printf("ERROR: Principal set is NULL!\\n");
}else {
       printf("Principal set size %d.\\n", principal_set->size);
       for (i = 0; i < principal_set->size; i++) {
           printf("Principal[%d] = %s.\\n", i, principal_set->strings[i]);
       }
       am_string_set_destroy(principal_set);
}

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

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

/* refresh token, idle time should be 0 after refresh */
status = am_sso_refresh_token(sso_handle);
if (status != AM_SUCCESS) {
    printf("Error refreshing token !\\n");
    return 1;
}
printf("After refresh, idle time is %d.\\n", am_sso_get_idle_time(sso_handle));

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

/* we are done with sso token handle. free memory for sso handle. */
status = am_sso_destroy_sso_token_handle(sso_handle);
if (status != AM_SUCCESS) {
```

```
        printf("Failed to free sso token handle.\\n");
        return 1;
    }

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

Listening and Notification

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

Note – For more information, see "Single Sign-on Properties" on page 71 and "`<am_notify.h>`" on page 222.

```
void sample_listener_func(
                  am_sso_token_handle_t sso_token_handle,
                  const am_sso_token_event_type_t event_type,
                  const time_t event_time,
                  void *opaque)
    {
        if (sso_token_handle != NULL) {
            const char *sso_token_id = am_sso_get_sso_token_id(sso_token_handle);
            boolean_t is_valid = am_sso_is_valid_token(sso_token_handle);
            printf("sso token id is %s.\\n",
                   sso_token_id==NULL?"NULL":sso_token_id);
            printf("session state is %s.\\n",
                     is_valid == B_TRUE ? "valid":"invalid");
            printf("event type %d.\\n", event_type);
            printf("event time %d.\\n", event_time);
        }
        else {
            printf("Error: sso token handle is null!");
        }
        if (opaque)
            *(int *)opaque = 1;
        return;
    }
```

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

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

    /* initialize sso as in previous samples */

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

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

Non-Web Applications

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

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

Single Sign-on Data Types

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

- "`am_sso_token_handle_t`" on page 79
- "`am_sso_token_listener_func_t`" on page 80

`am_sso_token_handle_t`

A pointer to the session information object.

Syntax

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

Members

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

am_sso_token_listener_func_t

Listener function declaration.

Syntax

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

Members

`am_sso_token_listener_func_t` has the following members:

`sso_token_handle` Pointer to the session information object.

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

```
typedef enum {
    AM_SSO_TOKEN_EVENT_TYPE_UNKNOWN = 0,
    AM_SSO_TOKEN_EVENT_TYPE_IDLE_TIMEOUT = 1,
    AM_SSO_TOKEN_EVENT_TYPE_MAX_TIMEOUT = 2,
    AM_SSO_TOKEN_EVENT_TYPE_LOGOUT = 3,
    AM_SSO_TOKEN_EVENT_TYPE_DESTROY = 5
} am_sso_token_event_type_t;
```

`event_time` Takes a value based on the standard `time_t` data type that represents the time at which the change event occurred.

`*args` Pointer to application-defined parameters.

Single Sign-on Functions

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

- "am_sso_add_listener()" on page 81
- "am_sso_add_sso_token_listener()" on page 83
- "am_sso_create_sso_token_handle()" on page 85
- "am_sso_destroy_sso_token_handle()" on page 86
- "am_sso_get_auth_level()" on page 86
- "am_sso_get_auth_type()" on page 87
- "am_sso_get_host()" on page 87
- "am_sso_get_idle_time" on page 88
- "am_sso_get_max_idle_time()" on page 88
- "am_sso_get_max_session_time()" on page 89
- "am_sso_get_principal()" on page 89
- "am_sso_get_principal_set()" on page 90
- "am_sso_get_property()" on page 90
- "am_sso_get_sso_token_id()" on page 91
- "am_sso_get_time_left()" on page 91
- "am_sso_init()" on page 92
- "am_sso_invalidate_token()" on page 93
- "am_sso_is_valid_token()" on page 94
- "am_sso_refresh_token()" on page 94
- "am_sso_remove_listener()" on page 95
- "am_sso_remove_sso_token_listener()" on page 96
- "am_sso_set_property()" on page 97
- "am_sso_validate_token()" on page 98

Before invoking any of the single sign-on functions, `am_web_init()` should be invoked by passing the bootstrap property and the local configuration file as parameters, and `am_agent_initialized()` should be invoked by passing the address of the boolean variable as a parameter — implying that `am_web.h` should be included in the file.

am_sso_add_listener()

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

Note – `am_sso_add_listener()` will not be removed after it is called once like "am_sso_add_sso_token_listener()" on page 83.

Details

The caller must do one of the following:

- Provide a URL to this function.
- Enable notification and provider a valid notification URL in the agent configuration properties that is passed to `am_sso_init()`.

See "Listening and Notification" on page 78 for more information.

Syntax

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

Parameters

This function takes the following parameters:

`listener`

The listener as described in "`am_sso_token_listener_func_t`" on page 80.

Note – When the listener is called, updated session information from OpenSSO Enterprise is passed in a temporary `sso_token_handle`.

`args`

Pointer to application-defined arguments to pass to the listener.

`dispatch_to_sep_thread`

Takes one of the values based on the `boolean_t` (defined in the `<am_types.h>` header file) that indicates whether the listener function should be called in the calling thread or dispatched to a thread from the internal thread pool managed by the C SDK.

Note – Calling the listener in a thread from an internal thread pool allows `am_notify()` to return immediately upon parsing the notification message rather than waiting for the listener functions to finish before returning.

Returns

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

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

am_sso_add_sso_token_listener()

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

Note – `am_sso_add_sso_token_listener()` is removed from memory after it is called once, differentiating its functionality from "`am_sso_add_listener()`" on page 81.

Details

The caller must do one of the following:

- Provide a URL to this function.
- Enable notification and provider a valid notification URL in the agent configuration properties that is passed to `am_sso_init()`.

See "Listening and Notification" on page 78 for more information.

Syntax

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

Parameters

This function takes the following parameters:

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

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

`listener` The listener as described in "`am_sso_token_listener_func_t` on page 80.

Note – When the listener is called, updated session information from OpenSSO Enterprise is passed in a temporary `sso_token_handle`.

`args` Arguments to pass to the listener.

`dispatch_to_sep_thread` Takes one of the values based on the `boolean_t` (defined in the `<am_types.h>` header file) that indicates whether the listener function should be called in the calling thread or dispatched to a thread from the internal thread pool managed by the C SDK.

Note – Calling the listener in a thread from an internal thread pool allows `am_notify()` to return immediately upon parsing the notification message rather than waiting for the listener functions to finish before returning.

Returns

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

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

am_sso_create_sso_token_handle()

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

Details

For more information, see "Single Sign-on Token Handles" on page 75.

Syntax

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

Parameters

This function takes the following parameters:

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

Returns

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

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

am_sso_destroy_sso_token_handle()

Destroys the specified single sign-on token handle.

Details

am_sso_destroy_sso_token_handle() does not log out the user or invalidate the session. For more information, see "Single Sign-on Token Handles" on page 75.

Syntax

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

Parameters

This function takes the following parameter:

sso_token_handle	Pointer to a am_sso_token_handle_t type which will be destroyed.

Returns

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

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

am_sso_get_auth_level()

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

Syntax

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

Parameters

This function takes the following parameter:

`sso_token` Pointer to a `am_sso_token_handle_t` type.

Returns

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

am_sso_get_auth_type()

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

Syntax

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

Parameters

This function takes the following parameter:

`sso_token` Pointer to a `am_sso_token_handle_t` type.

Returns

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

am_sso_get_host()

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

Syntax

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

Parameters

This function takes the following parameter:

`sso_token` Pointer to a `am_sso_token_handle_t` type.

Returns

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

am_sso_get_idle_time

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

Syntax

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

Parameters

This function takes the following parameter:

`sso_token_handle` Pointer to a `am_sso_token_handle_t` type.

Returns

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

am_sso_get_max_idle_time()

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

Syntax

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

Parameters

This function takes the following parameters:

`sso_token` Pointer to a `am_sso_token_handle_t` type.

Returns

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

am_sso_get_max_session_time()

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

Syntax

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

Parameters

This function takes the following parameter:

`sso_token` Pointer to a `am_sso_token_handle_t` type.

Returns

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

am_sso_get_principal()

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

Syntax

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

This function takes the following parameter:

`sso_token` Pointer to a `am_sso_token_handle_t` type.

Returns

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

am_sso_get_principal_set()

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

Syntax

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

Parameters

This function takes the following parameter:

sso_token	Pointer to a am_sso_token_handle_t type.

Returns

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

am_sso_get_property()

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

Syntax

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

Parameters

This function takes the following parameters:

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

Returns

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

am_sso_get_sso_token_id()

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

Syntax

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

Parameters

This function takes the following parameters:

`sso_token_handle` Pointer to a `am_sso_token_handle_t` type.

Returns

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

am_sso_get_time_left()

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

Syntax

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

Parameters

This function takes the following parameters:

`sso_token_handle` Pointer to a `am_sso_token_handle_t` type.

Returns

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

am_sso_init()

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

Details

`am_sso_init()` takes as input a properties file that contains name/value pairs, and returns status on the success or failure of the initialization. This call must be made before calling any other `am_sso_*` functions. See "Single Sign-on Properties" on page 71 for more information.

Syntax

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

Parameters

This function takes the following parameter:

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

Note – See "`am_properties_t`" on page 139 for more information.

Returns

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

`AM_SUCCESS` If the initialization was successful.

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

am_sso_invalidate_token()

Invalidates or destroys the session on OpenSSO Enterprise associated with the single sign-on token handle.

Details

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

Syntax

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

Parameters

This function takes the following parameter:

`sso_token_handle` Pointer to a `am_sso_token_handle_t` type.

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

Returns

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

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

AM_FAILURE	If any other error occurred.

am_sso_is_valid_token()

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

Details

am_sso_is_valid_token() looks in the passed sso_token_handle to check for validity. It does *not* go to OpenSSO Enterprise.

Syntax

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

Parameters

This function takes the following parameter:

sso_token_handle	Pointer to a am_sso_token_handle_t type.

Returns

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

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

am_sso_refresh_token()

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

Details

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

Syntax

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

Parameters

This function takes the following parameter:

`sso_token_handle`	Pointer to a `am_sso_token_handle_t` type.

Returns

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

`AM_SUCCESS`	If the token was refreshed with no errors.
`AM_INVALID_ARGUMENT`	If the `sso_token_handle` parameter is invalid.
`AM_SERVICE_NOT_INITIALIZED`	If the Session Service was not initialized with `am_sso_init()`.
`AM_SERVICE_NOT_AVAILABLE`	If server returned service not available.
`AM_HTTP_ERROR`	If an HTTP error was encountered while communicating with OpenSSO Enterprise.
`AM_ERROR_PARSING_XML`	If an error occurred while parsing XML from OpenSSO Enterprise.
`AM_ACCESS_DENIED`	If access was denied while communicating with OpenSSO Enterprise.
`AM_SESSION_FAILURE`	If the session validation failed.
`AM_FAILURE`	If any other error occurred.

am_sso_remove_listener()

Removes a single sign-on token listener.

Details

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

Syntax

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

Parameters

This function takes the following parameter:

`listener` The listener as described in "am_sso_token_listener_func_t" on page 80.

Returns

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

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

am_sso_remove_sso_token_listener()

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

Details

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

Syntax

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

Parameters

This function takes the following parameters:

`sso_token_handle` Pointer to a `am_sso_token_handle_t` type.

`listener`	The listener as described in "`am_sso_token_listener_func_t`" on page 80.

Returns

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

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

am_sso_set_property()

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

Details

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

Syntax

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

Parameters

This function takes the following parameters:

`sso_token_handle`	Pointer to a `am_sso_token_handle_t` type.
`name`	Pointer to the name of the property.

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

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

Returns

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

`AM_SUCCESS`	If the property was successfully set.
`AM_INVALID_ARGUMENT`	If the `sso_token_handle` is invalid.
`AM_FAILURE`	If any other error occurred.

am_sso_validate_token()

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

Details

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

Syntax

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

Parameters

This function takes the following parameter:

`sso_token_handle`	Pointer to a `am_sso_token_handle_t` type.

Returns

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

`AM_SUCCESS`	If single sign-on token is valid. The information is updated.
`AM_INVALID_SESSION`	If the session is invalid. The information is updated.
`AM_INVALID_ARGUMENT`	If the input parameter is invalid.

AM_SERVICE_NOT_INITIALIZED	If Session Service is not initialized.
AM_SERVICE_NOT_AVAILABLE	If OpenSSO Enterprise returned service not available.
AM_HTTP_ERROR	If HTTP error encountered while communicating with OpenSSO Enterprise.
AM_ERROR_PARSING_XML	If error parsing XML from OpenSSO Enterprise.
AM_ACCESS_DENIED	If access is denied while communicating with OpenSSO Enterprise.
AM_FAILURE	If any other error occurred.

CHAPTER 5

Logging Data Types and Functions

Sun OpenSSO Enterprise contains public data types and functions you can use for logging on the local system or on Sun OpenSSO Enterprise. Reference summaries include a short description, syntax, parameters and returns. The code is contained in the `<am_log.h>` header file. The sample source `am_log_test.c` demonstrates the basic usage of the logging API to record information such as user activity, traffic patterns and authorization violations. This chapter contains the following sections:

- "The Logging API for C" on page 101
- "Logging Data Types" on page 102
- "Logging Functions" on page 103

The Logging API for C

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

- The `am_log_init()` function by the application must be called before using any other `am_log_*` interfaces. If either the SSO, authentication, or policy initialization functions (`am_sso_init()`, `am_auth_init()`, or `am_policy_init()`) are called, `am_log_init()` does not need to be called as each of the three aforementioned functions call `am_log_init()` internally.
- The **`am_log_record_*`** interfaces can be used to set or update information in the log record. They include:
  - "`am_log_record_add_loginfo()`" on page 109

- "am_log_record_create()" on page 110
- "am_log_record_destroy()" on page 111
- "am_log_record_populate()" on page 112
- "am_log_record_set_log_level()" on page 112
- "am_log_record_set_log_message()" on page 113
- "am_log_record_set_loginfo_props()" on page 114

- The following are convenience functions that provide simplified access to existing log records. They include:
  - "am_log_record_set_log_level()" on page 112
  - "am_log_record_set_log_message()" on page 113

Logging Data Types

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

- "am_log_record_t" on page 102
- "am_log_module_id_t" on page 102

am_log_record_t

Represents the information and message values to be recorded.

Note – See Chapter 15, "Recording Events with the Logging Service," in *Sun OpenSSO Enterprise 8.0 Technical Overview* for information regarding events that are logged and the log fields to which they are written.

Syntax

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

Members

am_log_record is an opaque structure with no accessible members.

am_log_module_id_t

Represents the identifier for a logging module.

Note – See "am_log_add_module()" on page 103 for information on logging modules.

Syntax

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

Members

am_log_module_id_t has no members.

Logging Functions

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

- "am_log_add_module()" on page 103
- "am_log_flush_remote_log()" on page 105
- "am_log_init()" on page 105
- "am_log_is_level_enabled()" on page 106
- "am_log_log()" on page 107
- "am_log_log_record()" on page 108
- "am_log_record_add_loginfo()" on page 109
- "am_log_record_create()" on page 110
- "am_log_record_destroy()" on page 111
- "am_log_record_populate()" on page 112
- "am_log_record_set_log_level()" on page 112
- "am_log_record_set_log_message()" on page 113
- "am_log_record_set_loginfo_props()" on page 114
- "am_log_set_levels_from_string()" on page 115
- "am_log_set_log_file()" on page 115
- "am_log_set_module_level()" on page 116
- "am_log_set_remote_info()" on page 117
- "am_log_vlog()" on page 118

am_log_add_module()

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

Details

The currently used *module* file names are:

- AuthService
- NamingService
- PolicyService
- SessionService
- PolicyEngine
- ServiceEngine
- Notification
- PolicyAgent
- RemoteLog
- all

Syntax

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

Parameters

This function takes the following parameters:

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

Returns

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

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

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

am_log_flush_remote_log()

Flushes all the log records in the OpenSSO Enterprise log buffer.

Syntax

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

Parameters

This function takes no parameters:

Returns

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

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

am_log_init()

Initializes the Logging Service.

Details

`am_log_init()` writes events to the logs on the OpenSSO Enterprise server.

Syntax

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

Parameters

This function takes the following parameter:

`log_init_params`	Properties used to initialize the Logging Service.

Note – See "`am_properties_t`" on page 139 for more information.

Returns

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

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

am_log_is_level_enabled()

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

Details

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

Syntax

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

Parameters

This function takes the following parameters:

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

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

Returns

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

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

`!0` If a message will be generated.

`0` If a message will not be generated.

am_log_log()

Produces a logging message from the specified event string.

Details

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

- The message is produced only if the level defined for the specified module is greater than or equal to the level defined for the message. See "am_log_is_level_enabled()" on page 106.
- `am_log_log()` directly enumerates arguments for the `format` string. See "am_log_vlog()" on page 118 for another method.

Syntax

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

Parameters

This function takes the following parameters:

`moduleID` The identifier of the OpenSSO Enterprise logging module to which the message is relevant.

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

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

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

Note – The set of additional arguments needed by `format` are either enumerated directly or passed using the standard `va_list` mechanism as appropriate to the call. See "`am_log_vlog()`" on page 118.

Returns

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

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

`!0` If the message is logged.

`0` If the message will not be logged.

am_log_log_record()

Writes the given log record to the specified logging module.

Syntax

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

Parameters

This function takes the following parameters:

`record`	The log record pointer.
`log_name`	Pointer to the name of the logging module to which the log record will be written.
`logged_by_token_id`	Pointer to a valid `SSOTokenID` identifying the user to whom the log record applies.

Note – This is required to access the Logging Service on OpenSSO Enterprise.

Returns

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

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

am_log_record_add_loginfo()

Updates a log record with additional information.

Syntax

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

Parameters

This function takes the following parameters:

`record`	A log record pointer.
`key`	Pointer to the log field being updated.

Note – See Chapter 15, "Recording Events with the Logging Service," in *Sun OpenSSO Enterprise 8.0 Technical Overview* for information regarding events that are logged and the log fields to which they are written.

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

Returns

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

am_log_record_create()

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

Syntax

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

Parameters

This function takes the following parameters:

`record_ptr` Pointer to a log record pointer.

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

```
typedef enum am_log_record_log_level {

    /* Log Level as defined by JDK 1.4 */

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

```
    AM_LOG_LEVEL_FINEST = 300,

    /* Log Levels defined by Access Manager */

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

} am_log_record_log_level_t;
```

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

Returns

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

am_log_record_destroy()

Destroys the specified log record.

Syntax

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

Parameters

This function takes the following parameter:

`record` A log record pointer.

Returns

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

am_log_record_populate()

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

Details

See "Single Sign-on Token Handles" on page 75 for more information.

Syntax

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

Parameters

This function takes the following parameters:

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

Returns

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

am_log_record_set_log_level()

Sets the level for the specified log record.

Syntax

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

Parameters

This function takes the following parameters:

record	A log record pointer.

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

```
typedef enum am_log_record_log_level {

    /* Log Level as defined by JDK 1.4 */

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

    /* Log Levels defined by Access Manager */

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

} am_log_record_log_level_t;
```

Returns

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

am_log_record_set_log_message()

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

Syntax

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

Parameters

This function takes the following parameters:

`record` A log record pointer.

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

Returns

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

am_log_record_set_loginfo_props()

Updates the specified log record with additional information.

Details

`log_info` is expected to have the information formatted as key/value pairs in a properties map. Delete the `am_properties_t` pointer only when finished with the SDK. See "`am_properties_t`" on page 139 for more information.

Syntax

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

Parameters

This function takes the following parameters:

`record` A log record pointer.

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

Returns

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

am_log_set_levels_from_string()

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

Details

The format of the string must be:

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

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

Syntax

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

Parameters

This function takes the following parameter:

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

Returns

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

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

am_log_set_log_file()

Sets the name of the file to use for logging.

Syntax

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

Parameters

This function takes the following parameter:

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

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

Returns

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

`AM_SUCCESS` If the logging file is successfully set.

`AM_NO_MEMORY` If unable to allocate memory for internal data structures.

`AM_FAILURE` If an error of any type occurs.

am_log_set_module_level()

Sets the level for the specified logging module.

Syntax

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

Parameters

This function takes the following parameters:

`moduleID` The identifier of the logging module.

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

```
typedef enum am_log_level {
    AM_LOG_ALWAYS = -1, /* always logged */
    AM_LOG_NONE,    /* never logged, typically used to turn off a module */
    AM_LOG_ERROR,    /* used for error messages */
    AM_LOG_WARNING,    /* used for warning messages */
    AM_LOG_INFO,    /* used for informational messages */
```

```
    AM_LOG_DEBUG,    /* used for debug messages */
    AM_LOG_MAX_DEBUG,    /* used for more detailed debug messages */
    AM_LOG_AUTH_REMOTE = 128, /* logged deny and/or allow */
    AM_LOG_AUTH_LOCAL = 256
} am_log_level_t;
```

Returns

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

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

am_log_set_remote_info()

Initializes the remote log service.

Details

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

Syntax

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

Parameters

This function takes the following parameters:

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

Note – `log_props` is expected to have the information formatted as a properties map in key/value pairs. Delete the `am_properties_t` pointer only when finished with the SDK. See "`am_properties_t`" on page 139 for more information.

Returns

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

am_log_vlog()

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

Details

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

- The message is produced only if the level defined for the specified module is greater than or equal to the level defined for the message. See "`am_log_is_level_enabled()`" on page 106.
- `am_log_vlog()` passes the standard `va_list` as an argument for the `format` string. See "`am_log_log()`" on page 107 for another method.

Syntax

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

Parameters

This function takes the following parameters:

`moduleID` The identifier of the OpenSSO Enterprise logging module to which the message is relevant.

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

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

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

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

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

Note – The set of additional arguments needed by `format` are either enumerated directly or passed using the standard `va_list` mechanism as appropriate to the call. See "`am_log_log()`" on page 107.

Returns

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

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

`!0` If the message can be logged.

`0` If the message will not be logged.

CHAPTER 6

Mapping Data Types and Functions

Sun OpenSSO Enterprise contains public data types and functions you can use for creating, destroying, and manipulating map objects. Reference summaries include a short description, syntax, parameters and returns. The code is contained in the `<am_map.h>` header file. The sample source `am_policy_test.c` demonstrates the basic usage of some of the basic mapping functions. This chapter contains the following sections:

- "The Mapping API for C" on page 121
- "Mapping Data Types" on page 121
- "Mapping Functions" on page 123

The Mapping API for C

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

Mapping Data Types

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

- "`am_map_t`" on page 121
- "`am_map_entry_iter_t`" on page 122
- "`am_map_value_iter_t`" on page 122

`am_map_t`

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

Syntax

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

Members

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

Memory Concerns

Free the allocated structure by calling `am_map_destroy()`. See "`am_map_destroy()`" on page 125.

am_map_entry_iter_t

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

Syntax

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

Members

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

am_map_value_iter_t

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

Syntax

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

Members

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

Mapping Functions

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

- "am_map_clear()" on page 123
- "am_map_copy()" on page 124
- "am_map_create()" on page 125
- "am_map_destroy()" on page 125
- "am_map_entry_iter_destroy()" on page 126
- "am_map_entry_iter_get_first_value()" on page 126
- "am_map_entry_iter_get_key()" on page 127
- "am_map_entry_iter_get_values()" on page 128
- "am_map_entry_iter_is_entry_valid()" on page 129
- "am_map_entry_iter_next()" on page 129
- "am_map_erase()" on page 130
- "am_map_find()" on page 131
- "am_map_find_first_value()" on page 131
- "am_map_for_each()" on page 132
- "am_map_get_entries()" on page 133
- "am_map_insert()" on page 134
- "am_map_size()" on page 135
- "am_map_value_iter_destroy()" on page 136
- "am_map_value_iter_get()" on page 136
- "am_map_value_iter_is_value_valid()" on page 137
- "am_map_value_iter_next()" on page 137

am_map_clear()

Erases all of the entries in the specified map object.

Syntax

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

Parameters

This function takes the following parameter:

`map` The map object.

Returns

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

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

am_map_copy()

Makes a copy of the specified map object.

Details

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

Syntax

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

Parameters

This function takes the following parameters:

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

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

Returns

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

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

Memory Concerns

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

am_map_create()

Creates a new, empty map object.

Details

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

Syntax

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

Parameters

This function takes the following parameter:

map_ptr	Pointer specifying the location of the new map object.

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

Returns

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

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

am_map_destroy()

Destroys the specified map object.

Syntax

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

Parameters

This function takes the following parameter:

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

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

Returns

This function does not return a value.

Memory Concerns

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

am_map_entry_iter_destroy()

Destroys the specified entry iterator.

Syntax

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

Parameters

This function takes the following parameter:

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

Returns

This function does not return a value.

am_map_entry_iter_get_first_value()

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

Syntax

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

Parameters

This function takes the following parameter:

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

Returns

This function returns one of the following:

`char *` Returns the first associated value of the specified key. The order of insertion into the map does not guarantee the value returned.

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

Memory Concerns

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

am_map_entry_iter_get_key()

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

Syntax

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

Parameters

This function takes the following parameters:

`entry_iter` The specified entry iterator.

Returns

This function returns one of the following values:

`char *` Returns the key.

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

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

am_map_entry_iter_get_values()

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

Syntax

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

Parameters

This function takes the following parameters:

`entry_iter` The specified entry iterator.

`value_iter_ptr` Pointer specifying the location of the value iterator.

Returns

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

`AM_SUCCESS` If no error was detected.

`AM_NO_MEMORY` If unable to allocate memory for the key iterator.

`AM_INVALID_ARGUMENT` If the `value_iter_ptr` argument is `NULL`.

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

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

Memory Concerns

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

am_map_entry_iter_is_entry_valid()

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

Syntax

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

Parameters

This function takes the following parameter:

`entry_iter` The specified entry iterator.

Returns

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

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

`!0` If the entry is valid.

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

am_map_entry_iter_next()

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

Syntax

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

Parameters

This function takes the following parameter:

`entry_iter` The specified event iterator.

Returns

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

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

`!0` If the entry is valid.

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

am_map_erase()

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

Syntax

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

Parameters

This function takes the following parameters:

`map` The specified map object.

`key` Pointer to the key of the entry to be erased.

Returns

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

`AM_SUCCESS` If the entry was successfully erased.

`AM_INVALID_ARGUMENT` If either the map or key argument is `NULL`.

`AM_NOT_FOUND` If the specified key is not in the map.

am_map_find()

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

Syntax

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

Parameters

This function takes the following parameters:

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

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

Returns

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

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

Memory Concerns

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

am_map_find_first_value()

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

Details

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

Syntax

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

Parameters

This function takes the following parameters:

`map` The specified map object.

`key` Pointer to a key.

Returns

This function returns one of the following values:

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

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

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

am_map_for_each()

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

Details

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

Syntax

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

```
                void **args),
                void **args);
```

Parameters

This function takes the following parameters:

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

Returns

This function returns one of the following values:

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

am_map_get_entries()

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

Details

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

Syntax

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

Parameters

This function takes the following parameters:

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

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

Returns

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

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

Memory Concerns

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

am_map_insert()

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

Details

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

Syntax

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

Parameters

This function takes the following parameters:

`map`	The specified map object.

`key` Pointer to the key for the entry.

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

`value` Pointer to the [new] value to be associated with the key.

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

Returns

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

`AM_SUCCESS` If the entry was successfully inserted into the map object.

`AM_NO_MEMORY` If unable to allocate memory for the value and, if necessary, the key.

`AM_INVALID_ARGUMENT` If either the map, key, or value argument is `NULL`.

am_map_size()

Returns the number of entries in the specified map object.

Syntax

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

Parameters

This function takes the following parameter:

`map` The specified map object.

Returns

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

am_map_value_iter_destroy()

Destroys the specified value iterator.

Details

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

Syntax

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

Parameters

This function takes the following parameter:

iter The specified value iterator.

Returns

This function does not return a value.

am_map_value_iter_get()

Returns the value currently referenced by the specified value iterator.

Syntax

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

Parameters

This function takes the following parameter:

iter The specified value iterator.

Returns

This function returns one of the following values:

`char *` The value.

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

am_map_value_iter_is_value_valid()

Determines if the specified value iterator references a valid value.

Syntax

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

Parameters

This function takes the following parameter:

`iter` The specified value iterator.

Returns

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

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

`!0` If the value is valid.

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

am_map_value_iter_next()

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

Syntax

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

Parameters

This function takes the following parameter:

`iter` The specified value iterator.

Returns

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

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

`!0` If successful.

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

CHAPTER 7

Property Data Types and Functions

Sun OpenSSO Enterprise contains public data types and functions you can use to associate properties between an external application and OpenSSO Enterprise itself. Reference summaries include a short description, syntax, parameters and returns. Prototypes for the types and functions are contained in the `<am_properties.h>` header file. This chapter contains the following sections:

- "Property Data Types" on page 139
- "Property Functions" on page 140

The Property API for C

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

Property Data Types

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

- "`am_properties_t`" on page 139
- "`am_properties_iter_t`" on page 140

`am_properties_t`

Pointer to a properties object.

Details

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

Syntax

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

Members

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

am_properties_iter_t

Pointer to the iterator for a properties object.

Syntax

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

Members

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

Property Functions

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

- "`am_properties_copy()`" on page 141
- "`am_properties_create()`" on page 142
- "`am_properties_destroy()`" on page 142
- "`am_properties_get()`" on page 143
- "`am_properties_get_boolean()`" on page 144
- "`am_properties_get_boolean_with_default()`" on page 145
- "`am_properties_get_entries()`" on page 145
- "`am_properties_get_positive_number()`" on page 146
- "`am_properties_get_signed()`" on page 147
- "`am_properties_get_signed_with_default()`" on page 148
- "`am_properties_get_unsigned()`" on page 148
- "`am_properties_get_unsigned_with_default()`" on page 149
- "`am_properties_get_with_default()`" on page 150

- "am_properties_is_set()" on page 151
- "am_properties_iter_destroy()" on page 152
- "am_properties_iter_get_key()" on page 152
- "am_properties_iter_get_value()" on page 153
- "am_properties_load()" on page 153
- "am_properties_set()" on page 154
- "am_properties_store()" on page 155

am_properties_copy()

Duplicates a specified properties object.

Details

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

Syntax

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

Parameters

This function takes the following parameters:

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

Returns

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

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

Memory Concerns

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

am_properties_create()

Creates an empty properties object.

Syntax

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

Parameters

This function takes the following parameters:

properties_ptr	Pointer to the location of the new properties object.

Returns

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

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

Memory Concerns

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

am_properties_destroy()

Destroys the specified properties object.

Details

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

Syntax

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

Parameters

This function takes the following parameter:

`properties` Pointer to the specified properties object.

Returns

This function returns no values.

am_properties_get()

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

Details

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

Syntax

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

Parameters

This function takes the following parameters:

`properties` Pointer to the specified properties object.

`key` Pointer to the specified key in the specified properties object.

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

Returns

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

`AM_SUCCESS` If no error is detected.

`AM_INVALID_ARGUMENT` If the `properties`, `key`, or `value_ptr` argument is `NULL`.

AM_NOT_FOUND	If the specified key has no associated value and a default value is not provided.
AM_INVALID_VALUE	If the value associated with the specified key cannot be parsed as required by the particular accessor function.
AM_NO_MEMORY	If insufficient memory is available to look up the key.

Memory Concerns

Do not modify value_ptr or free the memory.

am_properties_get_boolean()

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

Syntax

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

Parameters

This function takes the following parameters:

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

Returns

One of the following values stored in value_ptr:

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

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

am_properties_get_boolean_with_default()

Retrieves boolean type values from the specified properties object.

Details

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

Syntax

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

Parameters

This function takes the following parameters:

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

Returns

One of the following values stored in `value_ptr`:

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

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

am_properties_get_entries()

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

Syntax

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

Parameters

This function takes the following parameters:

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

Returns

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

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

am_properties_get_positive_number()

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

Syntax

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

Parameters

This function takes the following parameters:

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

Returns

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

am_properties_get_signed()

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

Syntax

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

Parameters

This function takes the following parameters:

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

Returns

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

am_properties_get_signed_with_default()

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

Details

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

Syntax

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

Parameters

This function takes the following parameters:

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

Returns

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

am_properties_get_unsigned()

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

Syntax

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

Parameters

This function takes the following parameters:

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

Returns

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

`am_properties_get_unsigned_with_default()`

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

Details

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

Syntax

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

Parameters

This function takes the following parameters:

`properties`	The specified properties object.

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

Returns

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

am_properties_get_with_default()

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

Details

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

Syntax

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

Parameters

This function takes the following parameters:

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

Returns

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

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

Memory Concerns

Do not modify value_ptr or free the memory.

am_properties_is_set()

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

Details

am_properties_is_set() does not return the value.

Syntax

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

Parameters

This function takes the following parameters:

properties	The specified properties object.
key	Pointer to the name of a key.

Returns

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

!0	If the property has a value.
0	Otherwise

am_properties_iter_destroy()

Destroys the specified iterator object.

Syntax

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

Parameters

This function takes the following parameter:

`properties_iter` The specified iterator object. It may be `NULL`.

Returns

This function returns no value.

am_properties_iter_get_key()

Returns the key of the entry currently referenced by the specified iterator object.

Syntax

```
#include "am_properties.h"
AM_EXPORT const char *
am_properties_iter_get_key (am_properties_iter_t properties_iter);
```

Parameters

This function takes the following parameter:

`properties_iter` The specified iterator object.

Returns

This function returns one of the following values:

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

`char *` The key.

am_properties_iter_get_value()

Returns the value of the key currently referenced by the specified iterator object.

Syntax

```
#include "am_properties.h"
AM_EXPORT const char *
am_properties_iter_get_value (am_properties_iter_t properties_iter);
```

Parameters

This function takes the following parameters:

`properties_iter` The specified iterator object.

Returns

This function returns one of the following values:

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

`char *` Value associated with the key.

am_properties_load()

Loads information from the specified properties file into the specified properties object.

Details

The file is assumed to follow the syntax of a standard Java properties file.

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_load(am_properties_t properties,
                   const char *file_name);
```

Parameters

This function takes the following parameters:

`properties` The specified properties object.

`file_name` Pointer to a properties file.

Returns

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

`AM_SUCCESS`	If no error has occurred.
`AM_NOT_FOUND`	If the specified file does not exist.
`AM_NSPR_ERROR`	If there is a problem accessing the file.
`AM_INVALID_ARGUMENT`	If `properties` or `file_name` is `NULL` or `file_name` points to an empty string.
`AM_NO_MEMORY`	If unable to allocate memory to store the property information.

am_properties_set()

Sets a value for the specified key in the specified properties object.

Details

The specified value will replace any existing value.

Syntax

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

Parameters

This function takes the following parameters:

`properties`	The specified properties object.
`key`	Pointer to the key being modified.
`value`	Pointer to the value to associate with the specified key.

Returns

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

`AM_SUCCESS`	If no error is detected.

AM_INVALID_ARGUMENT	If the properties, key, or value argument is NULL.
AM_NO_MEMORY	If unable to allocate memory to store the new key/value.

am_properties_store()

Retrieves key/value information from the specified properties object and stores it in the specified file.

Syntax

```
#include "am_properties.h"
AM_EXPORT am_status_t
am_properties_store(am_properties_t properties,
                    const char *file_name);
```

Parameters

This function takes the following parameters:

properties	The specified properties object.
file_name	Pointer to the file in which the property information will be stored.

Returns

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

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

CHAPTER 8

Web Agent Data Types and Functions

Sun OpenSSO Enterprise contains public data types and functions intended for use by Sun web agents. They can also be used to develop proprietary web agents. Reference summaries include a short description, syntax, parameters and returns. Prototypes for the types and functions are contained in the `<am_web.h>` header file. This chapter contains the following sections:

- "Web Agent API for C" on page 157
- "Web Agent Data Types" on page 158
- "Web Agent Function Pointers" on page 165
- "Web Agent Functions" on page 175

Web Agent API for C

The web agent application programming interface (API) for C are used by web agents to interact with OpenSSO Enterprise services such as the Authentication Service, the Session Service, the Policy Service, and the Logging Service. In order to use the data types and functions described herein, you should be familiar with web agents in general and how they work. The following books provide information for this purpose:

- *Sun OpenSSO Enterprise 8.0 Technical Overview*
- *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for Web Agents*
- *Sun OpenSSO Enterprise Policy Agent 3.0 Guide for Sun Java System Web Server 7.0*
- *Sun Java System Access Manager Policy Agent 2.2 User's Guide*
- *Sun Java System Access Manager Policy Agent 2.2 Guide for Sun Java System Web Server 6.1*

Web Agent Data Types

The web agent data types defined in <am_web.h> are:

- "am_web_add_header_in_response_t" on page 158
- "am_web_free_post_data_t" on page 158
- "am_web_get_post_data_t" on page 159
- "am_web_postcache_data_t" on page 160
- "am_web_render_result_t" on page 160
- "am_web_request_func_t" on page 161
- "am_web_request_params_t" on page 161
- "am_web_set_header_in_request_t" on page 163
- "am_web_set_method_t" on page 163
- "am_web_set_user_t" on page 164
- "post_urls_t" on page 164

am_web_add_header_in_response_t

Defines a data type for the am_web_add_header_in_response_func_t function pointer.

Details

See also "am_web_add_header_in_response_func_t" on page 165.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_add_header_in_response_func_t func;
    void **args;
} am_web_add_header_in_response_t;
```

Members

The structure has the following components:

func	Pointer to am_web_add_header_in_response_func_t function.
args	Pointer to a pointer to agent defined parameters.

am_web_free_post_data_t

Defines a data type for the am_web_free_post_data_t function pointer.

Details

See also "am_web_free_post_data_func_t" on page 166.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_free_post_data_func_t func;
    void **args;
} am_web_free_post_data_t;
```

Members

am_web_free_post_data_t has the following components:

func Pointer to am_web_free_post_data_func_t function.

args Pointer to a pointer to agent defined parameters.

am_web_get_post_data_t

Defines a data type for the am_web_get_post_data_t function pointer.

Details

See also "am_web_get_post_data_func_t" on page 168.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_get_post_data_func_t func;
    void **args;
} am_web_get_post_data_t;
```

Members

am_web_get_post_data_t has the following components:

func Pointer to am_web_get_post_data_func_t function.

args Pointer to a pointer to agent defined parameters.

am_web_postcache_data_t

Data type for temporarily storing POST data sent by the web agent to the OpenSSO Enterprise Session Service.

Details

Policy agents use the POST method to communicate with the Session Service. For information, see "Session Validation" in *Sun OpenSSO Enterprise 8.0 Technical Overview*.

Syntax

```
#include "am_web.h"
typedef struct am_web_postcache_data {
    char *value;
    char *url;
} am_web_postcache_data_t;
```

Members

`am_web_postcache_data_t` has the following components:

`value`	Pointer to the string value of the POST data.
`url`	Pointer to the destination URL of the POST.

am_web_render_result_t

Defines a data type for the `am_web_render_result_func_t` function pointer.

Details

See also "`am_web_render_result_func_t` on page 168.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_render_result_func_t func;
    void **args;
} am_web_render_result_t;
```

Members

`am_web_render_result_t` has the following components:

`func`	Pointer to the `am_web_render_result_func_t` function.
`args`	Pointer to a pointer to agent defined parameters.

am_web_request_func_t

Defines an all-inclusive data type for the function pointers used by the web agent.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_get_post_data_t get_post_data;
    am_web_free_post_data_t free_post_data;
    am_web_set_user_t set_user;
    am_web_set_method_t set_method;
    am_web_set_header_in_request_t set_header_in_request;
    am_web_add_header_in_response_t add_header_in_response;
    am_web_render_result_t render_result;
} am_web_request_func_t;
```

Members

`am_web_request_func_t` has the following components:

`get_post_data`	See "`am_web_get_post_data_t`" on page 159.
`free_post_data`	See "`am_web_free_post_data_t`" on page 158.
`set_user`	See "`am_web_set_user_t`" on page 164.
`set_method`	See "`am_web_set_method_t`" on page 163.
`set_header_in_request`	See "`am_web_set_header_in_request_t`" on page 163.
`add_header_in_response`	See "`am_web_add_header_in_response_t`" on page 158.
`render_result`	See "`am_web_render_result_t`" on page 160.

am_web_request_params_t

Represents the parameters of an HTTP request passed to a web server from a client browser.

Details

This structure represents the parameters of the HTTP request and includes `am_web_req_method_t` which defines the action to be performed on the resource (GET, POST, DELETE, etc.).

Syntax

```
#include "am_web.h"
typedef struct {
    char *url;                      /* The full request URL */
    char *query;                    /* query string if any */
    am_web_req_method_t method;     /* request method */
    char *path_info;                /* path info if any */
    char *client_ip;                /* client IP if any */
    char *cookie_header_val;        /* the cookie header value if any */
    void *reserved;                 /* reserved - do not set this */
} am_web_request_params_t;
```

Members

`am_web_request_params_t` has the following components:

`url` Pointer to the URL of the resource.

`query` The query string appended to the request URL, if any. For example, if the URL is `http://www.example.com?a=b&c=d`, the value of this parameter would be `a=b&c=d`.

`method` One of the following values of the `am_web_req_method_t` enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_WEB_REQUEST_UNKNOWN,
    AM_WEB_REQUEST_GET,
    AM_WEB_REQUEST_POST,
    AM_WEB_REQUEST_HEAD,
    AM_WEB_REQUEST_PUT,
    AM_WEB_REQUEST_DELETE,
    AM_WEB_REQUEST_TRACE,
    AM_WEB_REQUEST_OPTIONS
} am_web_req_method_t;
```

More information on these request methods can be found in `http://www.faqs.org/rfcs/rfc2068.html`.

`path_info` The path information in the request URL, if any.

`client_ip`	Pointer to the IP address from which the request was sent.
`cookie_header_val`	Pointer to the cookie header.
`reserved`	Do not set this.

am_web_set_header_in_request_t

Defines a data type for the `am_web_render_result_func_t` function pointer.

Details

See also "`am_web_render_result_func_t`" on page 168.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_set_header_in_request_func_t func;
    void **args;
} am_web_set_header_in_request_t;
```

Members

`am_web_set_header_in_request_t` has the following components:

`func`	Pointer to `am_web_set_header_in_request_func_t` function.
`args`	Pointer to a pointer to agent defined parameters.

am_web_set_method_t

Defines a data type for the `am_web_set_method_func_t` function pointer.

Details

See also "`am_web_set_method_func_t`" on page 173.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_set_method_func_t func;
    void **args;
} am_web_set_method_t;
```

Members

`am_web_set_method_t` has the following components:

`func` Pointer to the `am_web_set_method_func_t` function.

`args` Pointer to a pointer to agent defined parameters.

am_web_set_user_t

Defines a data type for the `am_web_set_user_func_t` function pointer.

Details

See also "am_web_set_user_func_t" on page 174.

Syntax

```
#include "am_web.h"
typedef struct {
    am_web_set_user_func_t func;
    void **args;
} am_web_set_user_t;
```

Members

`am_web_set_user_t` has the following components:

`func` Pointer to `am_web_set_user_func_t` function.

`args` Pointer to a pointer to agent defined parameters.

post_urls_t

A session information object defining the URLs used by the web agent to communicate with OpenSSO Enterprise.

Syntax

```
#include "am_web.h"
typedef struct post_urls {
    char *dummy_url;
    char *action_url;
    char *post_time_key;
} post_urls_t;
```

Members

`post_urls_t` has the following components:

`dummy_url` Pointer to a dummy URL to redirect for POST data preservation.

Note – *POST data preservation* is supported only on Policy Agent 2.2 for Sun Java System Web Server 6.1. The feature allows for submitted POST data to be preserved. See "Preserving POST Data on Sun Java System Web Server 7.0 Only" in *Sun Java System Access Manager Policy Agent 2.2 Guide for Sun Java System Web Server 6.1* for more information.

`action_url` Pointer to destination URL for a POST request.

`post_time_key` Pointer to a unique key used to tag a POST data entry.

Web Agent Function Pointers

The web agent function pointers must be written before calling the `am_web_process_request()` function to process a request. The function pointers defined in `<am_web.h>` are:

- "`am_web_add_header_in_response_func_t`" on page 165
- "`am_web_free_post_data_func_t`" on page 166
- "`am_web_get_cookie_sync_func_t`" on page 167
- "`am_web_get_post_data_func_t`" on page 168
- "`am_web_render_result_func_t`" on page 168
- "`am_web_result_set_header_func_t`" on page 169
- "`am_web_result_set_header_attr_in_request_func_t`" on page 170
- "`am_web_result_set_header_attr_in_response_func_t`" on page 171
- "`am_web_set_header_in_request_func_t`" on page 172
- "`am_web_set_method_func_t`" on page 173
- "`am_web_set_user_func_t`" on page 174

`am_web_add_header_in_response_func_t`

Adds (or sets) an HTTP header in a response.

Details

If a header of the same name already exists, it should be replaced with this header.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_add_header_in_response_func_t)
                              (void **args,
                               const char *name,
                               const char *val);
```

Parameters

This function takes the following parameter:

`args` Pointer to a pointer to agent defined parameters.

`name` Pointer to the name of the header.

`val` Pointer to the value of the header.

Returns

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

`AM_SUCCESS`	If the header was successfully added.
Appropriate `am_status_t` code	Otherwise.

am_web_free_post_data_func_t

Frees the data retrieved by `am_web_get_post_data_func_t`.

Details

The POST data can be `NULL` if it is not needed.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_free_post_data_func_t)
                              (void **args,
                               char *data);
```

Parameters

This function takes the following parameter:

`args` Pointer to a pointer to agent defined parameters.

`data` Pointer to the data.

Returns

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

`AM_SUCCESS` If the data was successfully freed.

Warning If not successfully freed, the status will be logged as a warning but ignored.

am_web_get_cookie_sync_func_t

Synchronizes two cookies.

Details

Currently, this is a dummy function. Do not use.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_get_cookie_sync_func_t)(
                                const char *cookieName,
                                char **dproCookie,
                                void **args);
```

Parameters

This function takes the following parameter:

`cookieName` Pointer to the cookie with which the OpenSSO Enterprise cookie will be synchronized.

`dproCookie` Pointer to a pointer to the OpenSSO Enterprise cookie.

`args` Pointer to a pointer to agent defined parameters.

Returns

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

`AM_SUCCESS` If the data was successfully freed.

Warning If not successfully freed, the status will be logged as a warning but ignored.

am_web_get_post_data_func_t

Retrieves post data.

Details

The returned POST data must be NULL terminated and will be freed by calling am_web_free_post_data_func_t.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_get_post_data_func_t)
                                (void **args,
                                 char **data);
```

Parameters

This function takes the following parameter:

args	Pointer to a pointer to agent defined parameters.
data	Pointer to a pointer to the data.

Returns

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

AM_SUCCESS	If the data was successfully retrieved.
HTTP internal error	Otherwise

am_web_render_result_func_t

Renders an HTML page based on the result of a web agent's enforcement.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_render_result_func_t)
                                (void **args,
                                 am_web_result_t http_result,
                                 char *data);
```

Parameters

This function takes the following parameter:

`args` Pointer to a pointer to agent defined parameters.

`http_result` One of the following values of the `am_web_result_t` enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_WEB_RESULT_OK,              /* access check was OK */
    AM_WEB_RESULT_OK_DONE,         /* OK and handled (for ex. notification) */
    AM_WEB_RESULT_FORBIDDEN,       /* access forbidden */
    AM_WEB_RESULT_REDIRECT,        /* redirected */
    AM_WEB_RESULT_ERROR            /* internal error */
} am_web_result_t;
```

For `AM_WEB_RESULT_OK_DONE`, the web agent should return an HTTP status code `200 OK` and the body of the HTTP response should be set to the string in the `data` parameter. For `AM_WEB_RESULT_REDIRECT`, the web agent should return an HTTP status code `302` and the Location header should be set to the redirect URL in the `data` argument. More information on these request methods can be found in `http://www.faqs.org/rfcs/rfc2068.html`.

`data` Pointer to a string defining user data, a URL, or other data.

Returns

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

`AM_SUCCESS` If page was successfully rendered.

Warning If not successfully freed, the status will be logged as a warning but ignored.

am_web_result_set_header_func_t

Sets LDAP attributes in an HTTP header.

Details

This function will be called when `com.sun.am.policy.agents.config.profile.attribute.fetch.mode` in the agent configuration properties is set to `HTTP_HEADER`. This property specifies if additional user profile attributes should be introduced into the request. Possible values are:

- `NONE`

- HTTP_HEADER
- HTTP_COOKIE

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_result_set_header_func_t)(
                                const char *key,
                                const char *attrValues,
                                void **args);
```

Parameters

This function takes the following parameter:

key	Pointer to the key to which the value will be set.
attrValues	Pointer to a string representing the values to be set.
args	Pointer to a pointer to agent defined parameters.

Returns

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

AM_SUCCESS	If the header was successfully set.

am_web_result_set_header_attr_in_request_func_t

Sets LDAP attributes defined in the request's HTTP cookie header.

Details

This function will be called when com.sun.am.policy.agents.config.profile.attribute.fetch.mode in the agent configuration properties is set to HTTP_COOKIE. This property specifies if additional user profile attributes should be introduced into the request. Possible values are:

- NONE
- HTTP_HEADER
- HTTP_COOKIE

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_result_set_header_attr_in_request_func_t)(
                                const char *cookieValues,
                                void **args);
```

Parameters

This function takes the following parameter:

`cookieValues`	Pointer to string representing the values in the cookie.
`args`	Pointer to a pointer to agent defined parameters.

Returns

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

`AM_SUCCESS`	If the header was successfully set.

am_web_result_set_header_attr_in_response_func_t

Sets LDAP attributes defined in the response's HTTP cookie header.

Details

This function will be called when `com.sun.am.policy.agents.config.profile.attribute.fetch.mode` in the agent configuration properties is set to `HTTP_COOKIE`. This property specifies if additional user profile attributes should be introduced into the request. Possible values are:

- `NONE`
- `HTTP_HEADER`
- `HTTP_COOKIE`

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_result_set_header_attr_in_response_func_t)(
                    const char *cookieValues,
                    void **args);
```

Parameters

This function takes the following parameter:

`cookieValues` Pointer to string representing the values in the cookie.

`args` Pointer to a pointer to agent defined parameters.

Returns

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

`AM_SUCCESS` If the header was successfully set.

am_web_set_header_in_request_func_t

Sets an HTTP header in a request.

Details

If a header of the same name already exists it should be replaced with this header.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_set_header_in_request_func_t)(
                    void **args,
                    const char *name,
                    const char *val);
```

Parameters

This function takes the following parameter:

`args` Pointer to a pointer to agent defined parameters.

`name` Pointer to the name of the header.

`val` Pointer to the value of the header.

Returns

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

`AM_SUCCESS` If header was successfully added.

Warning If not successfully freed, the status will be logged as a warning but ignored.

am_web_set_method_func_t

Sets the request method to be used by a web agent during cross domain single sign-on (CDSSO).

Details

In cases of CDSSO actions between OpenSSO Enterprise and web agents, the POST request method is used by the web agent. `am_web_set_method_func_t` is required to change the request method to POST, if necessary, from the method defined in the original HTTP request. See "Cross-Domain Single Sign-On Session" in *Sun OpenSSO Enterprise 8.0 Technical Overview* for additional information.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_set_method_func_t)(
                                void **args,
                                am_web_req_method_t method);
```

Parameters

This function takes the following parameter:

`args` Pointer to a pointer to agent defined parameters.

`method` One of the following values of the `am_web_req_method_t` enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_WEB_REQUEST_UNKNOWN,
    AM_WEB_REQUEST_GET,
    AM_WEB_REQUEST_POST,
    AM_WEB_REQUEST_HEAD,
    AM_WEB_REQUEST_PUT,
    AM_WEB_REQUEST_DELETE,
    AM_WEB_REQUEST_TRACE,
    AM_WEB_REQUEST_OPTIONS
} am_web_req_method_t;
```

More information on these request methods can be found in `http://www.faqs.org/rfcs/rfc2068.html`.

Returns

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

`AM_SUCCESS`	If the method was successfully set.
HTTP forbidden result	Otherwise.

am_web_set_user_func_t

Sets the user.

Details

The implementation code sets the user.

Syntax

```
#include "am_web.h"
typedef am_status_t (*am_web_set_user_func_t)(
                    void **args,
                    const char *user);
```

Parameters

This function takes the following parameter:

`args`	Pointer to a pointer to agent defined parameters.
`user`	Pointer to the user login.

Returns

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

`AM_SUCCESS`	If user was successfully set.
HTTP forbidden result	Otherwise.

Web Agent Functions

The web agent functions defined in <am_web.h> are:

- "am_web_init() for Agents 3.0" on page 176
- "am_agent_init()" on page 177
- "am_web_build_advice_response()" on page 177
- "am_web_check_cookie_in_post()" on page 178
- "am_web_check_cookie_in_query()" on page 179
- "am_web_clean_post_urls()" on page 181
- "am_web_cleanup()" on page 181
- "am_web_clear_attributes_map()" on page 182
- "am_web_create_post_page()" on page 182
- "am_web_create_post_preserve_urls()" on page 183
- "am_web_delete_agent_configuration()" on page 184
- "am_web_do_cookie_domain_set()" on page 184
- "am_web_do_cookies_reset()" on page 185
- "am_web_free_memory()" on page 186
- "am_web_get_agent_server_host()" on page 186
- "am_web_get_agent_configuration()" on page 187
- "am_web_get_agent_server_port()" on page 187
- "am_web_get_authType()" on page 188
- "am_web_get_cookie_name()" on page 188
- "am_web_get_notification_url()" on page 189
- "am_web_get_parameter_value()" on page 189
- "am_web_get_request_url()" on page 190
- "am_web_get_token_from_assertion()" on page 191
- "am_web_get_url_to_redirect()" on page 192
- "am_web_get_token_from_assertion()" on page 191
- "am_web_handle_notification()" on page 194
- "am_web_http_decode()" on page 195
- "am_web_is_access_allowed()" on page 195
- "am_web_is_cdsso_enabled()" on page 197
- "am_web_is_cookie_present()" on page 197
- "am_web_is_debug_on()" on page 198
- "am_web_is_in_not_enforced_ip_list()" on page 199
- "am_web_is_in_not_enforced_list()" on page 200
- "am_web_is_logout_url()" on page 200
- "am_web_is_max_debug_on()" on page 201
- "am_web_is_notification()" on page 202
- "am_web_is_owa_enabled()" on page 202
- "am_web_is_owa_enabled_change_protocol()" on page 203
- "am_web_is_owa_enabled_session_timeout_url()" on page 204
- "am_web_is_postpreserve_enabled()" on page 204

- "am_web_is_proxy_override_host_port_set()" on page 205
- "am_web_is_valid_fqdn_url()" on page 205
- "am_web_log_always()" on page 206
- "am_web_log_auth()" on page 207
- "am_web_log_debug()" on page 207
- "am_web_log_error()" on page 208
- "am_web_log_info()" on page 208
- "am_web_log_max_debug()" on page 209
- "am_web_log_warning()" on page 209
- "am_web_logout_cookies_reset()" on page 210
- "am_web_method_num_to_str()" on page 210
- "am_web_method_str_to_num()" on page 211
- "am_web_postcache_data_cleanup()" on page 212
- "am_web_postcache_insert()" on page 213
- "am_web_postcache_lookup()" on page 213
- "am_web_postcache_remove()" on page 214
- "am_web_process_request()" on page 215
- "am_web_remove_authnrequest()" on page 216
- "am_web_remove_parameter_from_query()" on page 217
- "am_web_result_attr_map_set()" on page 217
- "am_web_result_num_to_str()" on page 219
- "am_web_set_cookie()" on page 219

am_web_init() for Agents 3.0

Initializes the 3.0 web agent by loading the bootstrap properties file.

Syntax

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

Parameters

This function takes the following parameter:

agent_bootstrap_file	Pointer to the agent bootstrap file which resides on the machine local to the agent.
agent_config_file	Pointer to the agent configuration file which resides on the same machine as OpenSSO Enterprise.

Returns

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

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

am_agent_init()

Initializes an agent by creating the agent profile object and performing agent authentication to receive the initial agent configuration data from either OpenSSO Enterprise or the local configuration file.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_agent_init(boolean_t *pAgentAuthenticated);
```

Parameters

This function takes the following parameter:

`pAgentAuthenticated`	is the agent authenticated.

Returns

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

`AM_SUCCESS`	If the query parameter was found in the URL.
`AM_*`	If any other error occurred.

am_web_build_advice_response()

Builds an advice response.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_build_advice_response(const am_policy_result_t *policy_result,
                             const char *redirect_url,
```

```
                                char **advice_response);
```

Parameters

This function takes the following parameter:

`policy_result` Pointer to an `am_policy_result_t` data type.

Note – See "`am_policy_result_t` on page 51" for information.

`redirect_url` Pointer to a redirect URL.

`advice_response` Pointer to a pointer to the location of the advice response.

Returns

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

`AM_SUCCESS` If the query parameter was found in the URL.

`AM_*` If any other error occurred.

am_web_check_cookie_in_post()

Retrieves a user's `SSOToken`.

Details

In cases of cross domain single sign-on, OpenSSO Enterprise sends out a user's `SSOToken` using POST. This method uses POST to retrieve the `SSOToken` and set it in the foreign domain.

Syntax

```
#include "am_web.h"
M_WEB_EXPORT am_status_t
am_web_check_cookie_in_post(void ** args,
                            char ** dpro_cookie,
                            char ** request_url,
                            char **orig_req,
                            char *method,
                            char *response,
                            boolean_t responseIsCookie,
                 am_status_t (*set_cookie)(const char *, void **),
                        void (*set_method)(void **, char *),
```

```
                    void* agent_config
);
```

Parameters

This function takes the following parameter:

`args`	Pointer to a pointer to agent defined parameters.
`dpro_cookie`	Pointer to a pointer to the OpenSSO Enterprise cookie.
`request_url`	Pointer to a pointer to the CDSSO URL.
`orig_req`	Pointer to a pointer to the original request method.
`method`	Pointer to the changed method name.
`response`	Pointer to the response which will hold the POST data.
`responseIsCookie`	Returns one of the following values of the `boolean_t` enumeration (defined in the `<am_types.h>` header file): `B_TRUE` If using Liberty Alliance Project specifications. `B_FALSE` If OpenSSO Enterprise POST data.
`set_cookie`	Function pointer used to set the cookie in the foreign domain.
`set_method`	Function pointer used to reset the original method in the request.
`agent_config`	An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`AM_SUCCESS`	If the query parameter was found in the URL.
`AM_*`	If any other error occurred.

am_web_check_cookie_in_query()

Retrieves the cookie from a query string.

Details

In older versions of this product, when performing CDSSO, the cookie was part of the query string.

Syntax

```
AM_WEB_EXPORT am_status_t
am_web_check_cookie_in_query(void **args,
                             char **dpro_cookie,
                             const char *query,
                             char **request_url,
                             char ** orig_req,
                             char *method,
                 am_status_t (*set_cookie)(const char *, void **),
                        void (*set_method)(void **, char *),
                        void* agent_config
);
```

Parameters

This function takes the following parameter:

`args`	Pointer to a pointer to agent defined parameters.
`dpro_cookie`	Pointer to a pointer to the OpenSSO Enterprise cookie.
`query`	Pointer to the query.
`request_url`	Pointer to a pointer to the CDSSO URL.
`orig_req`	Pointer to a pointer to the original request method.
`method`	Pointer to a pointer to the changed method name.
`set_cookie`	Function pointer used to set the cookie in the foreign domain.
`set_method`	Function pointer used to reset the original method in the request.
`agent_config`	An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`AM_SUCCESS`	If the query parameter was found in the URL.
`AM_*`	If any other error occurred.

am_web_clean_post_urls()

Cleans up a post_urls_t data type.

Details

See "post_urls_t" on page 164 for more information.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_clean_post_urls(post_urls_t *posturl_struct);
```

Parameters

This function takes the following parameter:

posturl_struct Pointer to post_urls_t data type.

Returns

This function returns no values.

am_web_cleanup()

Cleans up any memory called by the am_web_* functions.

Details

This should be called before a web agent exits.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_cleanup();
```

Parameters

This function does not take any parameters.

Returns

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

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

am_web_clear_attributes_map()

Clears the profile, session and response attributes maps.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void am_web_clear_attributes_map(am_policy_result_t *result);
```

Parameters

This function takes the following parameter:

`result`	Pointer to the `am_policy_result_t`

Returns

This function returns no values.

am_web_create_post_page()

Creates an HTML form that submits the POST data with invisible name/value pairs.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT char *
am_web_create_post_page(const char *key,
                        const char *postdata,
                        const char *actionurl,
                        void* agent_config);
```

Parameters

This function takes the following parameters:

`key`	Pointer to the unique key identifying a POST data entry. It is used to remove post data once the page is re-posted.
`postdata`	Pointer to a browser encoded string representing the POST data entry.
`actionurl`	Pointer to the POST destination URL.
`agent_config`	An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

This function returns `char *` as the POST form to be submitted.

am_web_create_post_preserve_urls()

Constructs a `post_urls_t` data type during preservation of POST data.

Details

A `post_urls_t` data type contains a dummy POST URL, an action URL and a unique key. The dummy URL is filtered by the Server Application Function (SAF) to identify POST preservation redirects from general redirects. All three of these variables are required for POST preservation.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT post_urls_t *
am_web_create_post_preserve_urls(const char *request_url,
                                 void* agent_config);
```

Parameters

This function takes the following parameter:

`request_url`	Pointer to the request URL for POST in the HTTP request.
`agent_config`	An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

This function returns a `post_urls_t` data type. See "`post_urls_t`" on page 164 for information.

am_web_delete_agent_configuration()

Deletes the referenced object from the agent configuration data stored in the agent configuration cache.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_delete_agent_configuration(void*);
```

Parameters

This function takes no parameters.

Returns

This function deletes the latest configuration data, referenced by an object for a particular agent, from the agent configuration cache.

am_web_do_cookie_domain_set()

Sets the OpenSSO Enterprise cookie (called `iPlanetDirectoryPro`) for each domain configured in the `com.sun.am.policy.agents.config.cookie.domain.list` property in the agent configuration properties.

Details

`am_web_do_cookie_domain_set()` builds the set-cookie header for each domain, and calls the callback function declared in `setFunc` to set the cookie. In CDSSO, the callback function is called by `am_web_check_cookie_in_query()` and `am_web_check_cookie_in_post()` to set the cookie in the response.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_do_cookie_domain_set(am_status_t (*setFunc)(
                                        const char *,
                                        void **),
                                        void **args,
                                        const char *cookie
                                        void* agent_config);
```

Parameters

This function takes the following parameters:

`setFunc`	Function pointer with which the user can define their own function for setting the cookie in the foreign domain. The implementation defines the parameters.
`args`	Pointer to a pointer to agent defined parameters.
`cookie`	Pointer to the cookie.
`agent_config`	An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

am_web_do_cookies_reset()

Resets the cookies in a response before redirecting it for authentication to the OpenSSO Enterprise login page.

Details

This function resets the cookies specified in the agent configuration properties and invokes the set action that the agent passes in for each of them. It is enabled by setting the following properties:

- `com.sun.am.policy.agents.config.cookie.reset.enable`: This property must be set to `true` if the web agent needs to reset cookies in the response before redirecting them to OpenSSO Enterprise for authentication. By default it is set to `false`.
- `com.sun.am.policy.agents.config.cookie.reset.list`: This property (used only if `com.sun.am.policy.agents.config.cookie.reset.enable` is enabled) contains a comma-separated list of cookies that need to be included in the response redirected to OpenSSO Enterprise.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_do_cookies_reset(am_status_t (*setFunc)(
                                    const char *,
                                    void **),
```

```
                                        void **args,
                                        void* agent_config);
```

Parameters

This function takes the following parameters:

`setFunc`	Function pointer with which the user can define their own function for setting the cookie in the foreign domain. The implementation defines the parameters.
`args`	Pointer to a pointer to agent defined parameters.
`agent_config`	An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

am_web_free_memory()

Frees memory previously allocated by a `am_web_*` routine.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_free_memory(void *memory);
```

Parameters

This function takes the following parameter:

`memory`	Pointer to the memory.

Returns

This function returns no value.

am_web_get_agent_server_host()

Retrieves the name of the server host for the agent.

Syntax

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

Parameters

`agent_config`	An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

This function returns then name of the server host.

am_web_get_agent_configuration()

Retrieves the latest configuration data from the agent configuration cache. The returned instance gets used to serve requests.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void*
am_web_get_agent_configuration();
```

Parameters

This function takes no parameters.

Returns

This function returns the latest configuration data, for a particular agent, from the agent configuration cache.

am_web_get_agent_server_port()

Retrieves the port used by the agent on the server host.

Syntax

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

Parameters

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

This function returns a port number.

am_web_get_authType()

Determines if the `auth-type` value `DSAME` should be replaced by `Basic` in the Policy Agent 2.2 for Microsoft IIS 6.0.

Details

The Policy Agent 2.2 for Microsoft IIS 6.0 is defined in `<am_web.h>` as having `auth-type` value equal to `DSAME`.

```
#define     AM_WEB_AUTH_TYPE_VALUE     "DSAME"
```

`DSAME` is a hard-coded value representing all OpenSSO Enterprise authentication types other than `Basic`.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT const char *
am_web_get_authType(void* agent_config);
```

Parameters

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

This function returns either `DSAME` or `Basic`.

am_web_get_cookie_name()

Retrieves the name of the OpenSSO Enterprise cookie.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT const char *
am_web_get_cookie_name(void* agent_config);
```

Parameters

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

This function returns the name of the OpenSSO Enterprise cookie.

am_web_get_notification_url()

Retrieves the URL of the OpenSSO Enterprise Notification Service.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT const char *
am_web_get_notification_url(void* agent_config);
```

Parameters

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

This function returns the URL of the OpenSSO Enterprise Notification Service.

am_web_get_parameter_value()

Gets the value of the specified parameter from the specified request URL.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_get_parameter_value(const char *inpQuery,
                           const char *param_name,
```

```
                                char **param_value);
```

Parameters

This function takes the following parameters:

`inpQuery`	Pointer to the request URL that holds the parameter.
`param_name`	Pointer to the name of the parameter.
`param_value`	Pointer to a pointer to be filled with the value of the `param_name` parameter, if found.

Returns

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

`AM_SUCCESS`	If the query parameter was found in the URL.
`AM_*`	If any other error occurred.

Memory Concerns

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

am_web_get_request_url()

Parses the host request header field for a server host name, port, protocol, query parameter, and URI to return the requested URL to the web agent.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_get_request_url(const char *host_hdr,
                       const char *protocol,
                       const char *hostname,
                       size_t port,
                       const char *uri,
                       const char *query,
                       char **req_url,
                       void* agent_config);
```

Parameters

This function takes the following parameters:

`host_hdr` Pointer to the host header string of the HTTP request as passed from the browser.

`protocol` Pointer to the protocol used by the web container of the resource being requested.

`hostname` Pointer to the name of the host on which the resource being requested.

`port` Value based on the `size_t` defined in the standard `<stddef.h>` header file that reflects the port number of the resource being requested.

`uri` Pointer to the URI of the HTTP request.

Note – Most URLs have this basic form: *protocol*://*server*:*port*/*request-URI*. The *request-URI* portion of the URL is used by the web server to identify the document.

`query` The query string appended to the request URL, if any. For example, if the URL is `http://www.example.com?a=b&c=d`, the value of this parameter would be `a=b&c=d`.

`req_url` Pointer to a pointer to the OUT parameter to be populated with the value of the URL string to be used by the agent.

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`AM_SUCCESS` If the query parameter was found in the URL.

`AM_*` If any other error occurred.

am_web_get_token_from_assertion()

Returns the single sign-on token from the specified Security Assertion Markup Language (SAML) assertion.

Details

`am_web_get_token_from_assertion()` is used to retrieve the cookie sent by OpenSSO Enterprise in a SAML assertion.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_get_token_from_assertion(char *assertion,
                                char **token,
                                void* agent_config);
```

Parameters

This function takes the following parameters:

`assertion`	Pointer to the SAML assertion as an XML string.
`token`	Pointer to a pointer containing the single sign-on token identifier.
`agent_config`	An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`AM_SUCCESS`	If successful.
`AM_*`	Otherwise.

Memory Concerns

The returned identifier should be freed using `am_web_free()`.

am_web_get_url_to_redirect()

Returns a string representing a login URL or access denied URL to which the web agent should redirect.

Caution – `am_web_get_redirect_url()` has been deprecated and must not be used. It is supported for backward compatibility only.

Details

`am_web_get_url_to_redirect()` may redirect the user to the login URL or the access denied URL. The URL is appropriate to the provided status code and advice map returned by the Policy SDK. If redirecting to the login URL, the URL will include existing information specified in the URL from the configuration file (for example, the organization name) as well as the specified

goto parameter value which will be used by OpenSSO Enterprise after the user has successfully authenticated. The last parameter reserved must be passed with NULL.

Note – If the URL returned is not NULL, the caller of this function must call am_web_free_memory(void *) to free the pointer.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_get_url_to_redirect(am_status_t status,
                           const am_map_t advice_map,
                           const char *goto_url,
                           const char* method,
                           void *reserved,
                           char ** redirect_url,
                           void* agent_config);
```

Parameters

This function takes the following parameters:

status — The status from am_web_is_access_allowed().

Note – See "am_web_is_access_allowed()" on page 195.

---

advice_map — Any advice map from policy evaluation results.

goto_url — Pointer to the original URL which the user attempted to access.

method — Pointer to the original HTTP method: GET or POST.

reserved — This parameter is not currently used.

redirect_url — Pointer to a pointer containing the resulting OpenSSO Enterprise redirect URL.

agent_config — An agent configuration instance returned by am_web_get_agent_configuration(). This parameter should not be NULL.

Returns

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

AM_SUCCESS — If successful.

`AM_*`	Otherwise.

am_web_get_user_id_param()

Returns the value of the logged-in user.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT char * am_web_get_user_id_param(void* agent_config);
```

Parameters

This function takes the following parameters:

`agent_config`	An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

This function returns the value of the logged-in user.

am_web_handle_notification()

Handles notification data received by a web agent.

Details

`am_web_handle_notification()` generates logging messages for the event and any error that may occur during the processing of the notification.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_handle_notification(const char *data,
                           size_t data_length,
                           void* agent_config);
```

Parameters

This function takes the following parameters:

`data`	Pointer to the notification message as an XML string.

`data_length` Value based on the `size_t` defined in the standard `<stddef.h>` header file that reflects the length of the notification message.

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

This function returns no value.

am_web_http_decode()

URL decodes the specified URL encoded string.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT char *
am_web_http_decode(const char *string,
                   size_t len);
```

Parameters

This function takes the following parameters:

`string` Pointer to the URL encoded string.

`len` Value based on the `size_t` defined in the standard `<stddef.h>` header file that reflects the length of the string.

Returns

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

Memory Concerns

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

am_web_is_access_allowed()

Evaluates the access control policies for a specified web resource and action against those for a specified user.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_is_access_allowed(const char *sso_token,
                         const char *url,
                         const char *path_info,
                         const char *action_name,
                         const char *client_ip,
                         const am_map_t env_parameter_map,
                         am_policy_result_t *result,
                         void* agent_config);
```

Parameters

This function takes the following parameters:

`sso_token`	Pointer to the session token from the OpenSSO Enterprise cookie. This parameter may be `NULL` if there is no cookie present.
`url`	Pointer to the web resource URL. This parameter may not be `NULL`.
`path_info`	Pointer to the path information in the web resource URL, if any.
`action_name`	Pointer to the action (GET, POST, etc.) being performed on the specified resource URL. This parameter may not be `NULL`.
`client_ip`	Pointer to the IP address of the client attempting to access the specified resource URL. If client IP validation is turned on, this parameter may not be `NULL`.
`env_parameter_map`	A map object containing additional information about the user attempting to access the specified resource URL. This parameter may not be `NULL`.
`advices_map_ptr`	An output parameter where the `am_map_t` can be stored if the policy evaluation produces any advice information. This parameter may not be `NULL`. See "`am_map_t` on page 121" for more information.
`result`	Pointer to a policy result object.
`agent_config`	An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

AM_SUCCESS	If the evaluation was performed successfully and access is allowed.
AM_NO_MEMORY	If the evaluation was not successfully completed due to insufficient memory being available.
AM_INVALID_ARGUMENT	If any of the url, action_name, env_parameter_map, or advices_map_ptr parameters is NULL or if client IP validation is enabled and the client_ip parameter is NULL.
AM_INVALID_SESSION	If the specified session token does not refer to a currently valid session
AM_ACCESS_DENIED	If the policy information indicates that the user does not have permission to access the specified resource or any error is detected other than the ones listed above.

am_web_is_cdsso_enabled()

Returns a boolean specifying whether cross domain single sign-on is enabled in the agent's configuration file.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_cdsso_enabled(void* agent_config);
```

Parameters

agent_config	An agent configuration instance returned by am_web_get_agent_configuration(). This parameter should not be NULL.

Returns

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

B_TRUE	If cross domain single sign-on is enabled.
B_FALSE	Otherwise.

am_web_is_cookie_present()

Detects whether a cookie is present.

Details

This function will most probably be invoked in a loop. A cookie name and value is passed and the implementation checks whether the cookie is already listed. If not, the new cookie name and value are appended. If present, the value of the cookie name is updated.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT int
am_web_is_cookie_present(const char *cookie,
                         const char *value,
                         char **new_cookie);
```

Parameters

This function takes the following parameters:

`cookie`	Pointer to a cookie.
`value`	Pointer to a value.
`new_cookie`	Pointer to a pointer to the location of the new cookie.

Returns

This function returns one of the following integers as defined in `<am_web.h>`:

```
#define AM_WEB_COOKIE_EXIST             2
#define AM_WEB_COOKIE_MODIFIED          1
#define AM_WEB_COOKIE_ABSENT            0
#define AM_WEB_COOKIE_ERROR            -1
```

am_web_is_debug_on()

Returns a boolean specifying whether debug is enabled.

Details

`am_web_is_debug_on()` specifies whether the log level is set to anything greater than 0.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_debug_on();
```

Parameters

This function takes no parameters.

Returns

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

`B_TRUE` If the log level is set to anything greater than 0.

`B_FALSE` Otherwise.

am_web_is_in_not_enforced_ip_list()

Returns a boolean specifying whether the given IP address is defined as one where no authentication or authorization is required.

Details

IP addresses that are not enforced are defined in the `com.sun.am.policy.agents.config.notenforced_client_ip_list` property in the agent configuration properties. If the IP address from where the request was issued is not enforced, the request goes through without authentication or authorization.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_in_not_enforced_ip_list(const char *ip,
                                  void* agent_config);
```

Parameters

This function takes the following parameter:

`ip` Pointer to the IP address.

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`B_TRUE` If the IP is in the not enforced IP address list.

`B_FALSE` Otherwise.

am_web_is_in_not_enforced_list()

Returns a boolean specifying whether the URL being accessed by the user is in the not enforced list.

Details

URLs that are not enforced are defined in the `com.sun.am.policy.agents.config.notenforced_list` property in the agent configuration properties. If the URL is not enforced, the request goes through without authentication or authorization.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_in_not_enforced_list(const char *url,
                               const char *path_info,
                               void* agent_config);
```

Parameters

This function takes the following parameters:

`url` Pointer to the URL being accessed.

`path_info` Pointer to the path information in the URL being accessed, if any.

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`B_TRUE` If the URL is in the not enforced list.

`B_FALSE` Otherwise.

am_web_is_logout_url()

Returns a boolean specifying whether the specified URL is a logout URL.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_logout_url(const char *url,
                     void* agent_config);
```

Parameters

This function takes the following parameter:

`url`	Pointer to a URL.
`agent_config`	An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`B_TRUE`	If the specified URL is a logout URL.
`B_FALSE`	Otherwise.

am_web_is_max_debug_on()

Returns a boolean specifying whether the log level is set to 5.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_max_debug_on();
```

Parameters

This function takes no parameters.

Returns

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

`B_TRUE`	If the log level is set to 5.
`B_FALSE`	Otherwise.

am_web_is_notification()

Returns a boolean specifying whether the given URL is the Notification Service URL for the web agent as configured in the agent configuration properties.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_notification(const char *request_url
                       void* agent_config);
```

Parameters

This function takes the following parameter:

`request_url`	Pointer to the Notification Service URL
`agent_config`	An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`B_TRUE`	If the URL is the Notification Service URL of the agent as set in the agent configuration proeprties.
`B_FALSE`	Otherwise.

am_web_is_owa_enabled()

Returns the value of the `com.sun.identity.agents.config.iis.owa.enable` property.

Note – Outlook Web Access (OWA) is a web mail service installed on Microsoft Internet Information Services 6.0. OWA interacts with the Exchange server to retrieve e-mail, calender contacts, and the like. A policy agent can be used to provide SSO for OWA.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_owa_enabled(void* agent_config);
```

Parameters

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`B_TRUE` If OWA is enabled.

`B_FALSE` Otherwise.

am_web_is_owa_enabled_change_protocol()

Used to convert incoming `http` and `https` protocol to `https`. Returns the value of the `com.sun.identity.agents.config.iis.owa.enable.change.protocol` property. The property can have a value of either true or false; the default value is false.

Note – Outlook Web Access (OWA) is a web mail service installed on Microsoft Internet Information Services 6.0. OWA interacts with the Exchange server to retrieve email, calender contacts, and the like. A policy agent can be used to provide SSO for OWA.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_owa_enabled_change_protocol(void* agent_config);
```

Parameters

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`B_TRUE` If the protocol was changed.

`B_FALSE` Otherwise.

am_web_is_owa_enabled_session_timeout_url()

Returns the value of the `com.sun.identity.agents.config.iis.owa.enable.session_timeout_url` property, used to redirect to a custom session timeout page

---

Note – Outlook Web Access (OWA) is a web mail service installed on Microsoft Internet Information Services 6.0. OWA interacts with the Exchange server to retrieve email, calender contacts, and the like. A policy agent can be used to provide SSO for OWA.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT char *
am_web_is_owa_enabled_session_timeout_url(void* agent_config);
```

Parameters

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

This function returns the URL of the custom session timeout page.

am_web_is_postpreserve_enabled()

Returns a boolean specifying whether POST data preservation is enabled for clients in the agent configuration properties.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_postpreserve_enabled(void* agent_config);
```

Parameters

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`B_TRUE` If POST data preservation is on.

`B_FALSE` If POST data preservation is off.

am_web_is_proxy_override_host_port_set()

Determines if the `com.sun.am.policy.agents.config.override_host` and the `com.sun.am.policy.agents.config.override_port` properties are set for the proxy agent.

Details

These properties are defined in the agent configuration properties.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t am_web_is_proxy_override_host_port_set(void* agent_config);
```

Parameters

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`B_TRUE` If set.

`B_FALSE` Otherwise.

am_web_is_valid_fqdn_url()

Returns a boolean specifying whether the requested URL is a valid fully qualified domain name (FQDN) resource as configured in the agent configuration properties. For example, `myhost.mydomain.com`.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_is_valid_fqdn_url(const char *url,
                         void* agent_config);
```

Parameters

`url`	Pointer to a URL.
`agent_config`	An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`B_TRUE`	If the URL is using a fully qualified domain name.
`B_FALSE`	Otherwise.

am_web_log_always()

Log the given message regardless of the log level set in the agent configuration properties.

Syntax

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

Parameters

This function takes the following parameter:

`fmt`	Pointer to a formatted string message as in `printf`.

Returns

This function returns no values.

am_web_log_auth()

Log the given access allowed or denied message to the OpenSSO Enterprise logs.

Syntax

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

Parameters

This function takes the following parameters:

`access_type` One of the following values of the `am_web_access_t` enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_ACCESS_DENY = 0,
    AM_ACCESS_ALLOW
} am_web_access_t;
```

`fmt` Pointer to a formatted string message as in `printf`.

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`B_TRUE` If the call was successful.

`B_FALSE` Otherwise.

am_web_log_debug()

Log the given message at the debug level.

Syntax

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

Parameters

This function takes the following parameter:

`fmt` Pointer to a formatted string message as in `printf`.

Returns

This function returns no values.

am_web_log_error()

Log the given message at the debug log level.

Syntax

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

Parameters

This function takes the following parameter:

`fmt` Pointer to a formatted string message as in `printf`.

Returns

This function returns no values.

am_web_log_info()

Log the given message at the info log level.

Syntax

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

Parameters

This function takes the following parameter:

`fmt` Pointer to a formatted string message as in `printf`.

Returns

This function returns no values.

am_web_log_max_debug()

Log the given message at maximum debug level.

Syntax

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

Parameters

This function takes the following parameters:

`fmt` Pointer to a formatted string message as in `printf`.

Returns

This function returns no values.

am_web_log_warning()

Log the given message at the warning log level.

Syntax

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

Parameters

This function takes the following parameters:

`fmt` Pointer to a formatted string message as in `printf`.

Returns

This function returns no values.

am_web_logout_cookies_reset()

Resets cookie configured for reset on user logout.

Details

The reset function passed in is called for each cookie configured for reset. If the function failed for any cookie, the last failed status is returned.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_logout_cookies_reset(am_status_t (*setFunc)(
                                        const char *,
                                        void **),
                                        void **args,
                                        void* agent_config);
```

Parameters

This function takes the following parameters:

`setFunc`	Function pointer with which the user can define their own function for setting the cookie in the foreign domain. The implementation defines the parameters.
`args`	Pointer to a pointer to agent defined parameters.
`agent_config`	An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

This function returns no values.

am_web_method_num_to_str()

Converts a `am_web_req_method_t` number to a string.

Details

This function is used for logging the method in the local debug logs. It takes in a method name and returns a string value (such as GET, or POST).

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT const char *
am_web_method_num_to_str(am_web_req_method_t method);
```

Parameters

This function takes the following parameter:

`method` One of the following values of the `am_web_req_method_t` enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_WEB_REQUEST_UNKNOWN,
    AM_WEB_REQUEST_GET,
    AM_WEB_REQUEST_POST,
    AM_WEB_REQUEST_HEAD,
    AM_WEB_REQUEST_PUT,
    AM_WEB_REQUEST_DELETE,
    AM_WEB_REQUEST_TRACE,
    AM_WEB_REQUEST_OPTIONS
} am_web_req_method_t;
```

More information on these request methods can be found in http://www.faqs.org/rfcs/rfc2068.html.

Returns

This function returns a pointer to the string. If the number passed is not recognized, `UNKNOWN` is returned.

am_web_method_str_to_num()

Converts a `am_web_req_method_t` string to a number.

Details

This function does the opposite of the previously defined `am_web_method_num_to_str()`. See "`am_web_method_num_to_str()`" on page 210 for details.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT const char *
am_web_method_num_to_str(am_web_req_method_t method);
```

Parameters

This function takes the following parameter:

`method` One of the following values of the `am_web_req_method_t` enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_WEB_REQUEST_UNKNOWN,
    AM_WEB_REQUEST_GET,
    AM_WEB_REQUEST_POST,
    AM_WEB_REQUEST_HEAD,
    AM_WEB_REQUEST_PUT,
    AM_WEB_REQUEST_DELETE,
    AM_WEB_REQUEST_TRACE,
    AM_WEB_REQUEST_OPTIONS
} am_web_req_method_t;
```

More information on these request methods can be found in http://www.faqs.org/rfcs/rfc2068.html.

Returns

This function returns a pointer to the number If the string is not recognized, `AM_WEB_REQUEST_UNKNOWN` is returned.

am_web_postcache_data_cleanup()

Cleans up `am_web_postcache_data_t` data type.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_postcache_data_cleanup(am_web_postcache_data_t * const postentry_struct);
```

Parameters

This function takes the following parameter:

`postentry_struct` Pointer to `am_web_postcache_data_t` data type.

Returns

This function returns no value.

am_web_postcache_insert()

Inserts POST data entry in the POST cache.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_postcache_insert(const char *key,
                        const am_web_postcache_data_t *value,
                        void* agent_config);
```

Parameters

This function takes the following parameters:

`key` Pointer to the POST data preservation key for every entry.

`value` Pointer to the `am_web_postcache_data_t` data type.

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`B_TRUE` If the insertion was successful.

`B_FALSE` Otherwise.

am_web_postcache_lookup()

Looks up data in the POST cache.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT boolean_t
am_web_postcache_lookup(const char *key,
                        am_web_postcache_data_t *postdata_entry,
                        void* agent_config);
```

Parameters

This function takes the following parameters:

`key`	Pointer to the key to search POST data entry in POST data structure.
`postdata_entry`	Pointer to the `am_web_postcache_data_t` data type storing the POST data.
`agent_config`	An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`B_TRUE`	If the search was successful.
`B_FALSE`	Otherwise.

am_web_postcache_remove()

Removes data from the POST cache.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT void
am_web_postcache_remove(const char *key,
                        void* agent_config);
```

Parameters

This function takes the following parameter:

`key`	Pointer to the key of the entry to be removed.

agent_config	An agent configuration instance returned by am_web_get_agent_configuration(). This parameter should not be NULL.

Returns

This function returns no value.

am_web_process_request()

Processes a request access check and returns a HTTP result to be rendered by the agent.

Details

The render status is returned in the render_sts argument.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_web_result_t
am_web_process_request(am_web_request_params_t *req_params,
                       am_web_request_func_t *req_func,
                       am_status_t *render_sts,
                       void* agent_config);
```

Parameters

This function takes the following parameters:

req_params	Pointer to a am_web_request_params_t data type.
req_func	Pointer to a am_web_request_func_t data type.
render_sts	Pointer to one of the values of the am_status_t enumeration as defined in the <am_types.h> header file.
agent_config	An agent configuration instance returned by am_web_get_agent_configuration(). This parameter should not be NULL.

Returns

One of the following values of the am_web_result_t enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_WEB_RESULT_OK,               /* access check was OK */
    AM_WEB_RESULT_OK_DONE,          /* OK and handled (for ex. notification) */
```

```
    AM_WEB_RESULT_FORBIDDEN,        /* access forbidden */
    AM_WEB_RESULT_REDIRECT,         /* redirected */
    AM_WEB_RESULT_ERROR             /* internal error */
} am_web_result_t;
```

am_web_remove_authnrequest()

Removes those extra parameters from an authenticated request.

Details

When a user returns from CDSSO authentication, the request contains a list of parameters that were added when the user was authenticated. `am_web_remove_authnrequest()` removes these extra parameters so the request is forwarded to applications as it originally came from the browser.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_remove_authnrequest(char* inpString,
                           char **outString );
```

Parameters

This function takes the following parameters:

`inpString` Pointer to the URL received after CDSSO authentication.

`outString` Pointer to a pointer to the location of the new URL without the parameters.

Returns

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

`AM_SUCCESS` If successful.

`AM_*` The type of error indicated by the status value.

Memory Concerns

The value returned should be freed using `am_web_free()`.

am_web_remove_parameter_from_query()

Removes the given query parameter from the URL, if present.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_remove_parameter_from_query(const char* inpString,
                                   const char *remove_str,
                                   char **outString );
```

Parameters

This function takes the following parameters:

`inpString`	Pointer to the original URL.
`remove_str`	Pointer to the query parameter to be removed
`outString`	Pointer to a pointer to the location of the new URL without the query parameter.

Returns

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

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

Memory Concerns

The value returned should be freed using `am_web_free()`.

am_web_result_attr_map_set()

Processes `attr_response_map` from a `am_policy_result_t` data type and performs the set action.

Details

This function replaces `am_web_do_result_attr_map_set()` which is deprecated. It needs to be explicitly declare to use it. See `<am_web.h>` for more information.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_result_attr_map_set(am_policy_result_t *result,
                           am_web_result_set_header_func_t
                            setHeaderFunc,
                           am_web_result_set_header_attr_in_response_func_t
                            setCookieRespFunc,
                           am_web_result_set_header_attr_in_request_func_t
                            setCookieReqFunc,
                           am_web_get_cookie_sync_func_t getCookieSyncFunc,
                           void **args, void* agent_config);
```

Parameters

This function takes the following parameters:

`result` Pointer to the `am_policy_result_t`.

Note – See "`am_policy_result_t`" on page 51 for more information.

`setHeaderFunc` Pointer to the `am_web_result_set_header_func_t`.

`setCookieRespFunc` Pointer to the `am_web_result_set_header_attr_in_response_func_t`.

`setCookieReqFunc` Pointer to the `am_web_result_set_header_attr_in_request_func_t`.

`getCookieSyncFunc` Pointer to the `am_web_get_cookie_sync_func_t`.

`args` Pointer to a pointer to agent defined parameters.

`agent_config` An agent configuration instance returned by `am_web_get_agent_configuration()`. This parameter should not be `NULL`.

Returns

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

`AM_SUCCESS` If the call was successful.

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

Memory Concerns

The value returned should be freed using `am_web_free()`.

am_web_result_num_to_str()

Returns the name of a `am_web_result_t` as a string.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT const char *
am_web_result_num_to_str(am_web_result_t result);
```

Parameters

This function takes the following parameter:

`result` One of the following values of the `am_web_result_t` enumeration as defined:

```
#include "am_web.h"
typedef enum {
    AM_WEB_RESULT_OK,               /* access check was OK */
    AM_WEB_RESULT_OK_DONE,          /* OK and handled (for ex. notification) */
    AM_WEB_RESULT_FORBIDDEN,        /* access forbidden */
    AM_WEB_RESULT_REDIRECT,         /* redirected */
    AM_WEB_RESULT_ERROR             /* internal error */
} am_web_result_t;
```

Returns

This function returns a pointer to the string. For example, `AM_WEB_RESULT_OK` returns `AM_WEB_RESULT_OK`. If the result code passed is not recognized, `Unknown result code` is returned.

am_web_set_cookie()

Sets the specified cookie in the cookie header of the request.

Syntax

```
#include "am_web.h"
AM_WEB_EXPORT am_status_t
am_web_set_cookie(char *cookie_header,
```

```
                    const char *set_cookie_value,
                    char **new_cookie_header);
```

Parameters

This function takes the following parameters:

`cookie_header`	Pointer to the cookie header.
`set_cookie_value`	Pointer to the cookie name and value in the *set-cookie response header* form. This value should be the same as that of the `cookieValues` parameter of the `am_web_result_set_header_attr_in_request_func_t` function.
`new_cookie_header`	Pointer to a pointer to the original cookie header, or a new cookie header value which needs to be freed by the caller. This value can be `NULL`.

Returns

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

`AM_SUCCESS`	If successful.
`AM_*`	The type of error indicated by the status value.

CHAPTER 9

Additional Data Types and Functions

This chapter provides information and a reference guide to the data types and functions not documented elsewhere. This includes those described in the <am.h>, <am_notify.h>, <am_string_set.h>, <am_types.h>, and <am_util.h> header files. Reference summaries include a short description, syntax, parameters and returns. This chapter contains the following sections:

- "<am.h>" on page 221
- "<am_notify.h>" on page 222
- "<am_string_set.h>" on page 223
- "<am_types.h>" on page 225
- "<am_utils.h>" on page 228

<am.h>

<am.h> contains the am_cleanup() function which cleans up all internal data structures created by am_sso_init(), am_auth_init(), or am_policy_init(). It needs to be called only once at the end of any calls. After cleanup, the relevant initialize function must be called again before using any of its interfaces.

Note – Any properties passed to the initialization functions am_sso_init(), am_auth_init(), or am_policy_init() should be destroyed only after am_cleanup() is called.

---

am_cleanup() Syntax

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

am_cleanup() Parameters

This function takes no parameters.

am_cleanup() Returns

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

AM_SUCCESS	If successfully cleaned up.
AM_NSPR_ERROR	Netscape Portable Runtime (NSPR) error.
AM_FAILURE	If any other error occurred.

<am_notify.h>

<am_notify.h> contains the am_notify() function which parses and processes a session or policy notification message as an XML string. If the message is a session notification, any token handle listeners registered using am_sso_add_listener() will be called. If the message is a policy notification, the internal policy cache maintained by the policy module in the C SDK will be updated with the notification information only if the policy module has been initialized (using am_policy_init() and am_policy_service_init()).

am_notify() Syntax

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

am_notify() Parameters

This function takes the following parameters:

xmlmsg	Pointer to the XML message containing the notification.
policy_handle	Reference to the policy evaluation object.

am_notify() Returns

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

AM_SUCCESS	If the XML was successfully parsed and processed.
AM_INVALID_ARGUMENT	If any input parameter is invalid.
AM_ERROR_PARSING_XML	If an error occurred parsing the XML.
AM_ERROR_DISPATCH_LISTENER	If an error occurred dispatching the listener.
AM_FAILURE	If any other error occurred.

<am_string_set.h>

<am_string_set.h> contains public data types and functions intended to manipulate strings. The sample sources demonstrate basic usage of the string set API. This section contains the following sub sections:

- "String Data Types" on page 223
- "String Functions" on page 224

String Data Types

The string data type defined in <am_string_set.h> is am_string_set_t. The type holds a set of strings.

Details

am_string_set_allocate() and am_string_set_destroy() are used to allocate and free space for this type.

Syntax

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

Members

am_string_set_t has the following members:

size	Number of strings
strings	Pointer to a pointer to a list of strings.

String Functions

The string functions defined in <am_string_set.h> are:

- "am_string_set_allocate()" on page 224
- "am_string_set_destroy()" on page 224

am_string_set_allocate()

Allocates memory and initializes the string set size.

Syntax

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

Parameters

This function takes the following parameter:

size	Number of strings in the set.

Returns

This function returns the allocated am_string_set_t, or NULL if size is less than zero.

am_string_set_destroy()

Releases memory in the specified string set object by freeing each string in the set, then freeing the associated pointers, and finally, the structure itself.

Syntax

```
#include "am_string_set.h"
AM_EXPORT void
am_string_set_destroy(am_string_set_t *string_set);
```

Parameters

This function takes the following parameter:

`string_set` Pointer to the specified string set object.

Returns

This function returns no values.

<am_types.h>

<am_types.h> contains defined types and corresponding functions. They are:

- "boolean_t" on page 225
- "bool_t" on page 225
- "am_status_t" on page 225
- "am_status_to_string()" on page 226

boolean_t

Defines true or false boolean with the syntax:

```
#include "am_types.h"
typedef enum {
    B_FALSE,
    B_TRUE
} boolean_t;
```

bool_t

Defines true or false boolean with the syntax:

```
#include "am_types.h"
typedef enum {
    AM_FALSE = 0,
    AM_TRUE
} am_bool_t;
```

am_status_t

Defines error codes with the syntax:

```
#include "am_types.h"
typedef enum {
    AM_SUCCESS = 0,
    AM_FAILURE,
    AM_INIT_FAILURE,
    AM_AUTH_FAILURE,
    AM_NAMING_FAILURE,
    AM_SESSION_FAILURE,
    AM_POLICY_FAILURE,
    AM_NO_POLICY,
    AM_INVALID_ARGUMENT,
    AM_INVALID_VALUE,
    AM_NOT_FOUND,
    AM_NO_MEMORY,
    AM_NSPR_ERROR,
    AM_END_OF_FILE,
    AM_BUFFER_TOO_SMALL,
    AM_NO_SUCH_SERVICE_TYPE,
    AM_SERVICE_NOT_AVAILABLE,
    AM_ERROR_PARSING_XML,
    AM_INVALID_SESSION,
    AM_INVALID_ACTION_TYPE,
    AM_ACCESS_DENIED,
    AM_HTTP_ERROR,
    AM_INVALID_FQDN_ACCESS,
    AM_FEATURE_UNSUPPORTED,
    AM_AUTH_CTX_INIT_FAILURE,
    AM_SERVICE_NOT_INITIALIZED,
    AM_INVALID_RESOURCE_FORMAT,
    AM_NOTIF_NOT_ENABLED,
    AM_ERROR_DISPATCH_LISTENER,
    AM_REMOTE_LOG_FAILURE,
    AM_LOG_FAILURE,
    AM_REMOTE_LOG_NOT_INITIALIZED,
    AM_NUM_ERROR_CODES    /* This should always be the last. */
} am_status_t;
```

am_status_to_string()

Returns a message for the given error code.

Syntax

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

Parameters

This function takes the following parameter:

status	Given error code

Returns

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

AM_SUCCESS	Success.
AM_FAILURE	Failure.
AM_INIT_FAILURE	Initialization failure.
AM_AUTH_FAILURE	OpenSSO Enterprise Authentication Service failure.
AM_NAMING_FAILURE	OpenSSO Enterprise Naming Service failure.
AM_SESSION_FAILURE	OpenSSO Enterprise Session Service failure.
AM_POLICY_FAILURE	OpenSSO Enterprise Policy Service failure.
AM_NO_POLICY	No policy found.
AM_INVALID_ARGUMENT	Invalid argument.
AM_INVALID_VALUE	Invalid value.
AM_NOT_FOUND	OpenSSO Enterprise not found.
AM_NO_MEMORY	No memory.
AM_NSPR_ERROR	NSPR error.
AM_END_OF_FILE	Reached end of file.
AM_BUFFER_TOO_SMALL	If the defined size of the buffer is smaller than the encoded value.
AM_NO_SUCH_SERVICE_TYPE	No such service type found.
AM_SERVICE_NOT_AVAILABLE	Service is not available.
AM_ERROR_PARSING_XML	Error found during XML parsing.
XML AM_INVALID_SESSION	Invalid session.
AM_END_OF_FILE	Reached end of file.
AM_INVALID_ACTION_TYPE	Invalid action type.
AM_ACCESS_DENIED	Access denied.
AM_HTTP_ERROR	HTTP error.

AM_INVALID_FQDN_ACCESS	Invalid fully qualified domain name (FQDN) access.
AM_FEATURE_UNSUPPORTED	The feature or configuration is unsupported.
AM_AUTH_CTX_INIT_FAILURE	Authentication context initialization failed.
AM_SERVICE_NOT_INITIALIZED	Specified service was not initialized.
AM_INVALID_RESOURCE_FORMAT	Specified resource name does not follow the format required by the service.
AM_NOTIF_NOT_ENABLED	Notification Service is not enabled or no notification URL is set.
AM_ERROR_DISPATCH_LISTENER	Error occurred dispatching single sign-on (SSO) listener.
AM_REMOTE_LOG_FAILURE	Remote Logging Service encountered an error.
AM_LOG_FAILURE	Log encountered an error.
AM_REMOTE_LOG_NOT_INITIALIZED	Remote Logging Service is not initialized.

<am_utils.h>

<am_utils.h> contains functions to encode and decode cookies. The functions are:

- "am_http_cookie_encode()" on page 228
- "am_http_cookie_decode()" on page 229

am_http_cookie_encode()

Encodes an HTTP cookie.

Syntax

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

Parameters

This function takes the following parameters:

cookie	Pointer to the cookie.

buf	Pointer to the buffer where the encoded cookie will be stored.
len	The size of the buffer.

Returns

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

AM_SUCCESS	If the cookie was successfully encoded and stored.
AM_INVALID_ARGUMENT	If the cookie or buffer is NULL..
AM_BUFFER_TOO_SMALL	If the defined size is smaller than the encoded value.
AM_FAILURE	If any other error occurred.

am_http_cookie_decode()

Decodes an HTTP cookie.

Syntax

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

Parameters

This function takes the following parameters:

cookie	Pointer to the cookie.
buf	Pointer to the buffer where the encoded cookie will be stored.
len	The size of the buffer.

Returns

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

AM_SUCCESS	If the cookie was successfully decoded and coped.
AM_INVALID_ARGUMENT	If the cookie or buffer is NULL..
AM_BUFFER_TOO_SMALL	If the defined size is smaller than the decoded value.

AM_FAILURE	If any other error occurred.

Index

A

action, 60
action value, 49
advice messages, 51
advices, 51
agent properties, 52
am_agent_init(), 177
am_auth_abort(), 37
am_auth_callback_t, 28-29
am_auth_choice_callback_t, 30-31
am_auth_confirmation_callback_t, 32
am_auth_context_t, 27-28
am_auth_create_auth_context(), 37-38
am_auth_destroy_auth_context(), 38-39
am_auth_get_callback(), 39
am_auth_get_module_instance_names(), 40
am_auth_get_organization_name(), 41
am_auth_get_sso_token_id(), 41-42
am_auth_get_status(), 42-43
<am_auth.h>, 20
am_auth_has_more_requirements(), 43
am_auth_init(), 44
am_auth_language_callback_t, 33
am_auth_locale_t, 29-30
am_auth_login(), 44-45
am_auth_logout(), 45-46
am_auth_name_callback_t, 33-34
am_auth_num_callbacks(), 46
am_auth_password_callback_t, 34-35
am_auth_submit_requirements(), 46-47
am_auth_test.c, 21
am_auth_text_input_callback_t, 35
am_auth_text_output_callback_t, 36
am_cleanup(), 221-222
<am.h>, 20
am.h, 221-222
am_http_cookie_decode(), 229-230
am_http_cookie_encode(), 228-229
am_log_add_module(), 103-104
am_log_flush_remote_log(), 105
<am_log.h>, 20
am_log_init(), 105-106
am_log_is_level_enabled(), 106-107
am_log_log(), 107-108
am_log_log_record(), 108-109
am_log_module_id_t, 102-103
am_log_record_add_loginfo(), 109-110
am_log_record_create(), 110-111
am_log_record_destroy(), 111
am_log_record_populate(), 112
am_log_record_set_log_level(), 112-113
am_log_record_set_log_message(), 113-114
am_log_record_set_loginfo_props(), 114
am_log_record_t, 102
am_log_set_levels_from_string(), 115
am_log_set_log_file(), 115-116
am_log_set_module_level(), 116-117
am_log_set_remote_info(), 117-118
am_log_test.c, 21
am_log_vlog(), 118-119
am_map_clear(), 123-124
am_map_copy(), 124
am_map_create(), 125
am_map_destroy(), 125-126

am_map_entry_iter_destroy(), 126
am_map_entry_iter_get_first_value(), 126-127
am_map_entry_iter_get_key(), 127-128
am_map_entry_iter_get_values(), 128-129
am_map_entry_iter_is_entry_valid(), 129
am_map_entry_iter_next(), 129-130
am_map_entry_iter_t, 122
am_map_erase(), 130
am_map_find(), 131
am_map_find_first_value(), 131-132
am_map_for_each(), 132-133
am_map_get_entries(), 133-134
<am_map.h>, 20
am_map_insert(), 134-135
am_map_size(), 135
am_map_t, 121-122
am_map_value_iter_destroy(), 136
am_map_value_iter_get(), 136-137
am_map_value_iter_is_value_valid(), 137
am_map_value_iter_next(), 137-138
am_map_value_iter_t, 122
am_notify(), 222-223
<am_notify.h>, 20
am_notify.h, 222-223
am_policy_compare_urls(), 56-58
am_policy_destroy(), 58
am_policy_evaluate(), 58-60
am_policy_evaluate_ignore_url_notenforced(), 61-63
am_policy_get_url_resource_root(), 63-64
<am_policy.h>, 20
am_policy_init(), 64
am_policy_invalidate_session(), 64-65
am_policy_is_notification_enabled(), 65
am_policy_notify(), 66
am_policy_resource_canonicalize(), 66-67
am_policy_resource_has_patterns(), 67
am_policy_result_destroy(), 67-68
am_policy_result_t, 51-52
am_policy_service_init(), 68-69
am_policy_t, 53
am_policy_test.c, 21
am_properties_copy(), 141
am_properties_create(), 142
am_properties_destroy(), 142-143
am_properties_get(), 143-144
am_properties_get_boolean(), 144
am_properties_get_boolean_with_default(), 145
am_properties_get_entries(), 145-146
am_properties_get_positive_number(), 146-147
am_properties_get_signed(), 147
am_properties_get_signed_with_default(), 148
am_properties_get_unsigned(), 148-149
am_properties_get_unsigned_with_default(), 149-150
am_properties_get_with_default(), 150-151
<am_properties.h>, 20
am_properties_is_set(), 151
am_properties_iter_destroy(), 152
am_properties_iter_get_key(), 152
am_properties_iter_get_value(), 153
am_properties_iter_t, 140
am_properties_load(), 153-154
am_properties_set(), 154-155
am_properties_store(), 155
am_properties_t, 139-140
am_resource_traits_t, 53-55
am_sso_add_listener(), 81-83
am_sso_add_sso_token_listener(), 83-84
am_sso_create_sso_token_handle(), 85
am_sso_destroy_sso_token_handle(), 86
am_sso_get_auth_level(), 86-87
am_sso_get_auth_type(), 87
am_sso_get_idle_time(), 88
am_sso_get_max_idle_time(), 88-89
am_sso_get_max_session_time(), 89
am_sso_get_principal_set(), 90
am_sso_get_sso_token_id(), 91
am_sso_get_time_left(), 91-92
<am_sso.h>, 20
am_sso_init(), 92
am_sso_invalidate_token(), 93-94
am_sso_is_valid_token(), 94
am_sso_refresh_token(), 94-95
am_sso_remove_listener(), 95-96
am_sso_remove_sso_token_listener(), 96-97
am_sso_set_property(), 97-98
am_sso_test.c, 21
am_sso_token_event_type_t, 80
am_sso_token_handle_t, 79-80

am_sso_token_listener_func_t, 80
am_sso_validate_token(), 98-99
am_status_t, 225-226
am_status_to_string(), 226-228
am_string_set_allocate(), 224
am_string_set_destroy(), 224-225
<am_string_set.h>, 20
am_string_set.h, 223-225
am_string_set_t, 223-224
<am_types.h>, 20
am_types.h, 225-228
<am_utils.h>, 20
am_utils.h, 228-230
am_web_add_header_in_response_func_t, 165-166
am_web_add_header_in_response_t, 158
am_web_agent_test.c, 21
am_web_build_advice_response(), 177-178
am_web_check_cookie_in_post(), 178-179
am_web_check_cookie_in_query(), 179-180
am_web_clean_post_urls(), 181
am_web_cleanup(), 181-182
am_web_clear_attributes_map(), 182
am_web_create_post_page(), 182-183
am_web_create_post_preserve_urls(), 183
am_web_delete_agent_configuration(), 184
am_web_do_cookie_domain_set(), 184-185
am_web_do_cookies_reset(), 185-186
am_web_free_memory(), 186
am_web_free_post_data_func_t, 166-167
am_web_free_post_data_t, 158-159
am_web_get_agent_configuration(), 187
am_web_get_agent_server_host(), 186-187
am_web_get_agent_server_port(), 187-188
am_web_get_authType(), 188
am_web_get_cookie_name(), 188-189
am_web_get_cookie_sync_func_t, 167
am_web_get_notification_url(), 189
am_web_get_parameter_value(), 189-190
am_web_get_post_data_func_t, 168
am_web_get_post_data_t, 159
am_web_get_request_url(), 190-191
am_web_get_token_from_assertion(), 191-192
am_web_get_url_to_redirect(), 192-194
am_web_get_user_id_param(), 194
<am_web.h>, 20
am_web_handle_notification(), 194-195
am_web_http_decode(), 195
am_web_init() for Agents 3.0, 176-177
am_web_is_access_allowed(), 195-197
am_web_is_cdsso_enabled(), 197
am_web_is_cookie_present(), 197-198
am_web_is_debug_on(), 198-199
am_web_is_in_not_enforced_ip_list(), 199-200
am_web_is_in_not_enforced_list(), 200
am_web_is_logout_url(), 200-201
am_web_is_max_debug_on(), 201
am_web_is_notification(), 202
am_web_is_owa_enabled(), 202-203, 203
am_web_is_owa_enabled_session_timeout_url(), 204
am_web_is_postpreserve_enabled(), 204-205
am_web_is_proxy_override_host_port_set(), 205
am_web_is_valid_fqdn_url(), 205-206
am_web_log_always(), 206
am_web_log_auth(), 207
am_web_log_debug(), 207-208
am_web_log_error(), 208
am_web_log_info(), 208-209
am_web_log_max_debug(), 209
am_web_log_warning(), 209-210
am_web_logout_cookies_reset(), 210
am_web_method_num_to_str(), 210-211
am_web_method_str_to_num(), 211-212
am_web_postcache_data_cleanup(), 212-213
am_web_postcache_data_t, 160
am_web_postcache_insert(), 213
am_web_postcache_lookup(), 213-214
am_web_postcache_remove(), 214-215
am_web_process_request(), 215-216
am_web_remove_authnrequest(), 216
am_web_remove_parameter_from_query(), 217
am_web_render_result_func_t, 168-169
am_web_render_result_t, 160-161
am_web_request_func_t, 161
am_web_request_params_t, 161-163
am_web_result_attr_map_set(), 217-219
am_web_result_num_to_str(), 219
am_web_result_set_header_attr_in_request_func_t, 170-
am_web_result_set_header_attr_in_response_func_t, 171

am_web_result_set_header_func_t, 169-170
am_web_set_cookie(), 219-220
am_web_set_header_in_request_func_t, 172-173
am_web_set_header_in_request_t, 163
am_web_set_method_func_t, 173-174
am_web_set_method_t, 163-164
am_web_set_user_func_t, 174
am_web_set_user_t, 164
apache_agent.c, 21
authentication
  am_auth_abort(), 37
  am_auth_callback_t, 28-29
  am_auth_choice_callback_t, 30-31
  am_auth_confirmation_callback_t, 32
  am_auth_context_t, 27-28
  am_auth_create_auth_context(), 37-38
  am_auth_destroy_auth_context(), 38-39
  am_auth_get_callback(), 39
  am_auth_get_module_instance_names(), 40
  am_auth_get_organization_name(), 41
  am_auth_get_sso_token_id(), 41-42
  am_auth_get_status(), 42-43
  am_auth_has_more_requirements(), 43
  am_auth_init(), 44
  am_auth_language_callback_t, 33
  am_auth_locale_t, 29-30
  am_auth_login(), 44-45
  am_auth_logout(), 45-46
  am_auth_name_callback_t, 33-34
  am_auth_num_callbacks(), 46
  am_auth_password_callback_t, 34-35
  am_auth_submit_requirements(), 46-47
  am_auth_text_input_callback_t, 35
  am_auth_text_output_callback_t, 36
  am_cleanup(), 221-222
  C API, 25-27
authentication call sequence, 25-26
authentication callback data types, 30-36
authentication data types, 27-30
authentication data types and functions, 25-47
authentication functions,
  am_auth_text_input_callback_t, 36-47
authentication properties, 26-27

B

binary action value, 49
bool_t, 225
boolean action value, 49
boolean_t, 225

C

C API
  authentication, 25-27
  authentication call sequence, 25-26
  authentication callback data types, 30-36
  authentication data types, 27-30
  authentication functions, 36-47
  authentication properties, 26-27
  logging, 101-102
  policy, 49-50
  single sign-on, 71-79
  single sign-on calls, 73-79
  single sign-on properties, 71-73
  single sign-on token handles, 75-76
cleanup function, 221-222
cookie functions, 228-230
cookies
  decode, 228-230
  encode, 228-230

D

data types
  am_status_t, 225-226
  authentication, 27-30
  authentication callback, 30-36
  bool_t, 225
  boolean_t, 225
  mapping, 121-122
  policy, 51-55
  property, 139-140
  single sign-on, 79-80
  strings, 223-224
  web agents, 158-165
development, web policy agents, 21
documentation, 13-15

documentation *(Continued)*
OpenSSO Enterprise, 13-14
related products, 15

F

functions
am_http_cookie_decode(), 229-230
am_http_cookie_encode(), 228-229
am_status_to_string(), 226-228
authentication, 36-47
mapping, 123-138
property, 140-155
strings, 224-225

H

header files, 20

I

initialization and cleanup, single sign-on, 73-75

L

listening and notification, single sign-on, 78-79
logging
am_log_add_module(), 103-104
am_log_flush_remote_log(), 105
am_log_init(), 105-106
am_log_is_level_enabled(), 106-107
am_log_log(), 107-108
am_log_log_record(), 108-109
am_log_module_id_t, 102-103
am_log_record_add_loginfo(), 109-110
am_log_record_create(), 110-111
am_log_record_destroy(), 111
am_log_record_populate(), 112
am_log_record_set_log_level(), 112-113
am_log_record_set_log_message(), 113-114
am_log_record_set_loginfo_props(), 114

logging *(Continued)*
am_log_record_t, 102
am_log_set_levels_from_string(), 115
am_log_set_log_file(), 115-116
am_log_set_module_level(), 116-117
am_log_set_remote_info(), 117-118
am_log_vlog(), 118-119
overview, 101-102
logging data types and functions, 101-119

M

map, 121
mapping
am_map_clear(), 123-124
am_map_copy(), 124
am_map_create(), 125
am_map_destroy(), 125-126
am_map_entry_iter_destroy(), 126
am_map_entry_iter_get_first_value(), 126-127
am_map_entry_iter_get_key(), 127-128
am_map_entry_iter_get_values(), 128-129
am_map_entry_iter_is_entry_valid(), 129
am_map_entry_iter_next(), 129-130
am_map_entry_iter_t, 122
am_map_erase(), 130
am_map_find(), 131
am_map_find_first_value(), 131-132
am_map_for_each(), 132-133
am_map_get_entries(), 133-134
am_map_insert(), 134-135
am_map_size(), 135
am_map_t, 121-122
am_map_value_iter_destroy(), 136
am_map_value_iter_get(), 136-137
am_map_value_iter_is_value_valid(), 137
am_map_value_iter_next(), 137-138
am_map_value_iter_t, 122
mapping API for C, 121
data types, 121-122
functions, 123-138
mapping data types and functions, 121-138

N

non-web applications, single sign-on, 79
notify function, 222-223

O

overview
  logging API for C, 101-102
  policy API for C, 49-50

P

policy
  action, 60
  advice messages, 51
  am_cleanup(), 221-222
  am_notify(), 222-223
  am_policy_compare_urls(), 56-58
  am_policy_destroy(), 58
  am_policy_evaluate(), 58-60
  am_policy_evaluate_ignore_url_notenforced(), 61-63
  am_policy_get_url_resource_root(), 63-64
  am_policy_init(), 64
  am_policy_invalidate_session(), 64-65
  am_policy_is_notification_enabled(), 65
  am_policy_notify(), 66
  am_policy_resource_canonicalize(), 66-67
  am_policy_resource_has_patterns(), 67
  am_policy_result_destroy(), 67-68
  am_policy_result_t, 51-52
  am_policy_service_init(), 68-69
  am_policy_t, 53
  am_resource_traits_t, 53-55
  C API, 49-50
  evaluation, 50
  relative relationship, 50
  resource traits, 50
  resources strings, 50
  session token, 59, 61
policy data types, 51-55
policy data types and functions, 49-69
policy evaluation, 50
POST data preservation, 165
post_urls_t, 164-165
properties
  am_properties_copy(), 141
  am_properties_create(), 142
  am_properties_destroy(), 142-143
  am_properties_get(), 143-144
  am_properties_get_boolean(), 144
  am_properties_get_boolean_with_default(), 145
  am_properties_get_entries(), 145-146
  am_properties_get_positive_number(), 146-147
  am_properties_get_signed(), 147
  am_properties_get_signed_with_default(), 148
  am_properties_get_unsigned(), 148-149
  am_properties_get_unsigned_with_default(), 149-150
  am_properties_get_with_default(), 150-151
  am_properties_is_set(), 151
  am_properties_iter_destroy(), 152
  am_properties_iter_get_key(), 152
  am_properties_iter_get_value(), 153
  am_properties_load(), 153-154
  am_properties_set(), 154-155
  am_properties_store(), 155
  authentication, 26-27
  single sign-on, 71-73
property
  am_properties_iter_t, 140
  am_properties_t, 139-140
property API for C, 139
  data types, 139-140
  functions, 140-155
property data types and functions, 139-155

R

relative relationship, 50
required libraries, 22-23
  Linux, 22-23
  Solaris, 22
  Windows, 23
resource strings, 50
resource traits, 50
retrieving and setting properties, single sign-on, 76-78

S

samples, 21
session token, 59, 61
single sign-on
  am_cleanup(), 221-222
  am_notify(), 222-223
  am_sso_add_listener(), 81-83
  am_sso_add_sso_token_listener(), 83-84
  am_sso_create_sso_token_handle(), 85
  am_sso_destroy_sso_token_handle(), 86
  am_sso_get_auth_level(), 86-87
  am_sso_get_auth_type(), 87
  am_sso_get_idle_time(), 88
  am_sso_get_max_idle_time(), 88-89
  am_sso_get_max_session_time(), 89
  am_sso_get_principal_set(), 90
  am_sso_get_sso_token_id(), 91
  am_sso_get_time_left(), 91-92
  am_sso_init(), 92
  am_sso_invalidate_token(), 93-94
  am_sso_is_valid_token(), 94
  am_sso_refresh_token(), 94-95
  am_sso_remove_listener(), 95-96
  am_sso_remove_sso_token_listener(), 96-97
  am_sso_set_property(), 97-98
  am_sso_token_event_type_t, 80
  am_sso_token_handle_t, 79-80
  am_sso_token_listener_func_t, 80
  am_sso_validate_token(), 98-99
single sign-on C API, 71-79
  calls, 73-79
  initialization and cleanup, 73-75
  listening and notification, 78-79
  non-web applications, 79
  properties, 71-73
  retrieving and setting properties, 76-78
  token handles, 75-76
single sign-on calls, 73-79
single sign-on data types, 79-80
single sign-on data types and functions, 71-99
SSO, *See* single sign-on
SSOToken, 59, 61
SSOTokenID, 59, 61
string API for C, 223-225
string functions, 223-225
strings
  am_string_set_allocate(), 224
  am_string_set_destroy(), 224-225
  am_string_set_t, 223-224
  functions, 224-225

T

token handles, 75-76

W

web agent data types, 158-165
web agent data types and functions, 157-220
web agents
  am_agent_init(), 177
  am_web_add_header_in_response_func_t, 165-166
  am_web_add_header_in_response_t, 158
  am_web_build_advice_response(), 177-178
  am_web_check_cookie_in_post(), 178-179
  am_web_check_cookie_in_query(), 179-180
  am_web_clean_post_urls(), 181
  am_web_cleanup(), 181-182
  am_web_clear_attributes_map(), 182
  am_web_create_post_page(), 182-183
  am_web_create_post_preserve_urls(), 183
  am_web_delete_agent_configuration(), 184
  am_web_do_cookie_domain_set(), 184-185
  am_web_do_cookies_reset(), 185-186
  am_web_free_memory(), 186
  am_web_free_post_data_func_t, 166-167
  am_web_free_post_data_t, 158-159
  am_web_get_agent_configuration(), 187
  am_web_get_agent_server_host(), 186-187
  am_web_get_agent_server_port(), 187-188
  am_web_get_authType(), 188
  am_web_get_cookie_name(), 188-189
  am_web_get_cookie_sync_func_t, 167
  am_web_get_notification_url(), 189
  am_web_get_parameter_value(), 189-190
  am_web_get_post_data_func_t, 168
  am_web_get_post_data_t, 159

web agents *(Continued)*
am_web_get_request_url(), 190-191
am_web_get_token_from_assertion(), 191-192
am_web_get_url_to_redirect(), 192-194
am_web_get_user_id_param(), 194
am_web_handle_notification(), 194-195
am_web_http_decode(), 195
am_web_init() for Agents 3.0, 176-177
am_web_is_access_allowed(), 195-197
am_web_is_cdsso_enabled(), 197
am_web_is_cookie_present(), 197-198
am_web_is_debug_on(), 198-199
am_web_is_in_not_enforced_ip_list(), 199-200
am_web_is_in_not_enforced_list(), 200
am_web_is_logout_url(), 200-201
am_web_is_max_debug_on(), 201
am_web_is_notification(), 202
am_web_is_owa_enabled(), 202-203, 203
am_web_is_owa_enabled_session_timeout_url(), 204
am_web_is_postpreserve_enabled(), 204-205
am_web_is_proxy_override_host_port_set(), 205
am_web_is_valid_fqdn_url(), 205-206
am_web_log_always(), 206
am_web_log_auth(), 207
am_web_log_debug(), 207-208
am_web_log_error(), 208
am_web_log_info(), 208-209
am_web_log_max_debug(), 209
am_web_log_warning(), 209-210
am_web_logout_cookies_reset(), 210
am_web_method_num_to_str(), 210-211
am_web_method_str_to_num(), 211-212
am_web_postcache_data_cleanup(), 212-213
am_web_postcache_data_t, 160
am_web_postcache_insert(), 213
am_web_postcache_lookup(), 213-214
am_web_postcache_remove(), 214-215
am_web_process_request(), 215-216
am_web_remove_authnrequest(), 216
am_web_remove_parameter_from_query(), 217
am_web_render_result_func_t, 168-169
am_web_render_result_t, 160-161
am_web_request_func_t, 161
am_web_request_params_t, 161-163

web agents *(Continued)*
am_web_result_attr_map_set(), 217-219
am_web_result_num_to_str(), 219
am_web_result_set_header_attr_in_request_func_t, 170-171
am_web_result_set_header_attr_in_response_func_t, 171-17
am_web_result_set_header_func_t, 169-170
am_web_set_cookie(), 219-220
am_web_set_header_in_request_func_t, 172-173
am_web_set_header_in_request_t, 163
am_web_set_method_func_t, 173-174
am_web_set_method_t, 163-164
am_web_set_user_func_t, 174
am_web_set_user_t, 164
post_urls_t, 164-165
web policy agent development toolkit, 21